EEE102 INTRODUCTION TO DIGITAL CIRCUIT DESIGN

TERM PROJECT REPORT

WHACK A MOLE GAME WITH VGA

EFE TARHAN/22002840/SECTION 2/EEE
INSTRUCTOR : CEM TEKIN

INDEX

PURPOSE

Purpose of this experiment is to create a game project by using VGA connector. Developing and exhibiting digital design abilites is the main objective of term project of EEE102 course. Debugging during the coding and designing procedure by finding and using glitch removal circuit components or combinations is also an objective to be achieved. Understanding the mechanism behind VGA, being able to design a UI for an enjoyable and playable game, using sequential and combinatorial design elements to establish a glitchless game will be some of the challenges during the developement of the game.

DESIGN SPESIFICATIONS

In this project, following components and devices will be used ;

- A BASYS3 FPGA
- A monitor with VGA input
- A VGA connection cable
- A mini-breadboard
- A buzzer
- Jumper cables

Whack-A-Mole or Whac-A-Mole was a classical arcade game which is played with a plastic hammer. In order to get points in the game you must hit the head of the plastic moles that show up from holes. There is a time limit and the score you get is the number of moles that you have hit in one gameplay.

The game has slightly changed in the designed project. Main mission of the game is having as much as score as possible not in a restricted time but with minus 1 point for each of the errors. Moles will be changed with squares with different colors that blink when "moles" appear. To catch the moles the player will push the buttons. There will be 5 different colors located in the "+" shape on the screen.

The game will have two difficulty levels : easy and hard. Difficulty of the game will be set by changing the frequency of the random number generator. Change from easy to hard mode will be done by using a switch. Difficulty level will be displayed in the bottom left corner of the screen

Atari games usually have sound effects on their buttons which came from the built-in speakers which are located inside them. To make something similiar, a buzzer will be connected to BASYS3 and it will give a sound when a button is pressed. Sound will be turned on and of by using a switch and it will be displayed on the bottom right corner of the screen.

There will be a game clock on the screen which is going to indicate the time has passed since the start of the game. The clock will only show until 59 minutes 59 seconds then it will start counting from 00:00 again. The game clock will be designed by using clocks with various periods and a state machine dependent on those clocks. The digital clock will be located in the mid-bottom of the screen with white color.

On the top right corner of the screen, score of the player will be shown. The game will start from 10 points and end when user has no points left. Maximum score of the game is 99 points, after reaching 99 points it will start over counting from 0 but that won't mean the game has ended, it will mean that the user have one more life which is equivalent to 100 points.

A start screen and an ending screen will be designed by using different colored words. The game go to game screen from the beginning screen when the user pushes the down button and the game will start when user pushes the up button.

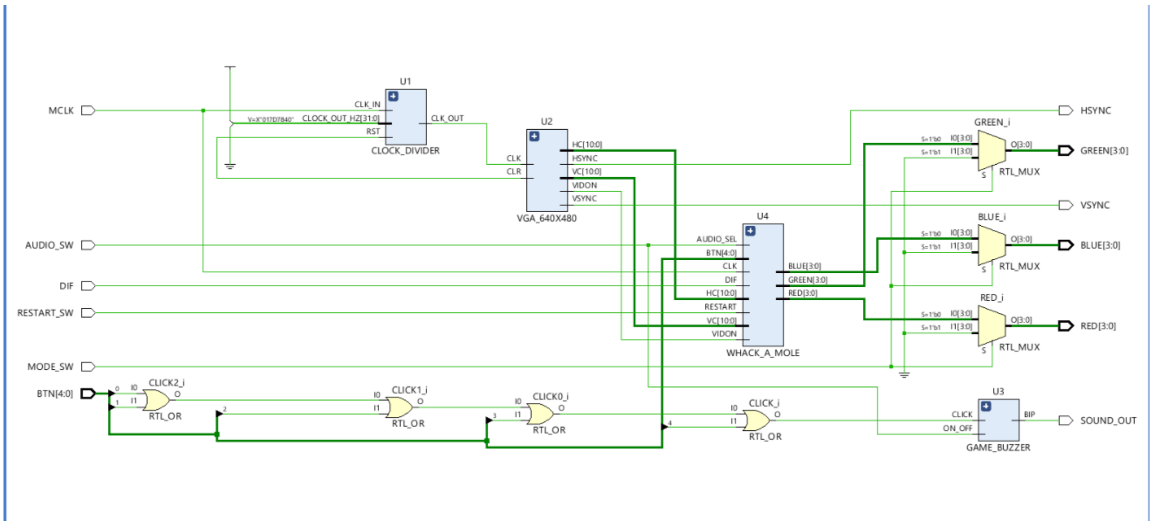
There will also be two switches for restarting the game and activating the screen saver, basically by completely painting the screen to black. The game will go on from where the player has left it when the screen is go on again.

METHODOLOGY

In this part of the report modules and their purposes will be explained by discussing the coded that has been used. Because some of the words that has used in the UI are whole modlues that consist of several submodules, therefore they won't be examined independently but main logic behind them will be explained. Same situation exist for two different clock divider modlules that has used in the project.

VGA_TOP

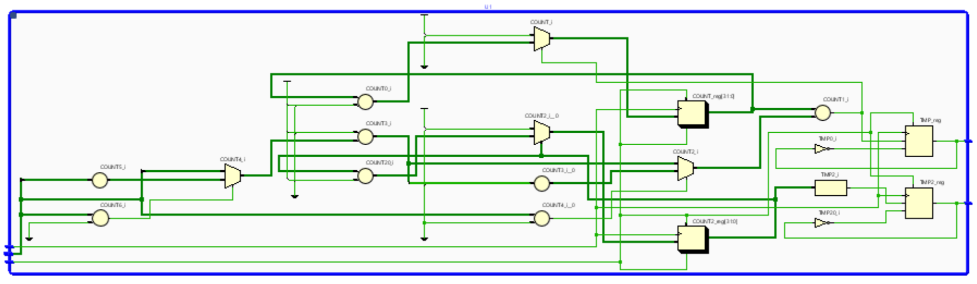
This is the main module where all other modules are connected to each other and also where the screen saver which is the blank black screen is set by a multiplexer between the R,G,B codes that consist of all R,G,B codes of the submodules and R,G,B codes consisted of x"0" values each indicating the black color. RTL schematic of this module can be seen below.



(Schematic 1: Schematic of VGA_TOP module)

CLOCK_DIVIDER and RANDOM_CLK_DIV

Those modules take built in clock of the BASYS3 as input which has 100Mhz frequency and generate desired clocks to make them available in modules to use them as module clocks. Schematic of the clock divider module can be seen below.

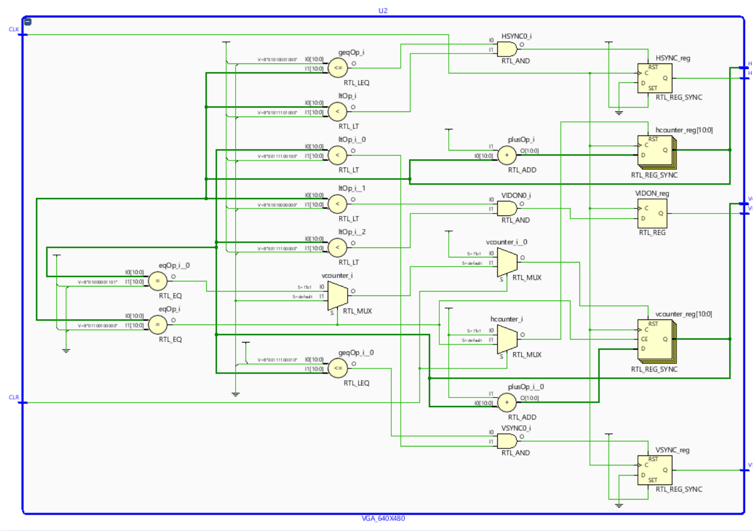


(Schematic 2: Schematic of the CLOCK_DIVIDER module)

VGA_640X480

This module creates the horizontal count, vertical count, hsync and vsync signals. VSYNC and HSYNC signals are input signals that go to the monitor. HC and VC signals help user to design the screen according to the current bit on the screen which is (VC,HC). The RGB screens also use the persistence of eye technology theory like we implemented in seven segment lab. The screen iterates thorugh all pixels on the screen in a time smaller than 1/60 seconds to create a image where humans can't understand the transition.

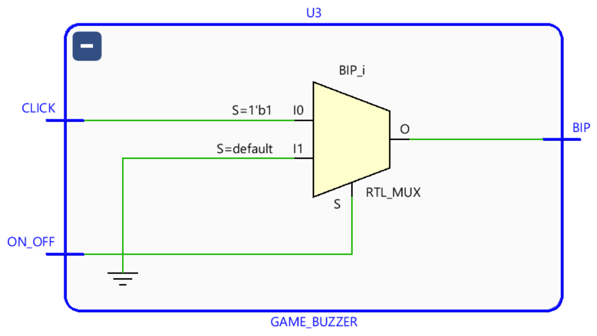
The VGA cable consists of 15 pins where one of the pin is GND, one is HSYNC, one is VSYNC and 3 signals for each R, G and B signals. The tone of the color is determined by the resistance in the cable corresponding to the bit of the signal. For example 4th bit of red signal is connected to monitor with 520 ohm resistance but 3rd bit is connected with 1Kohm resistance, 2nd is 2 Kohm and 1st is 4Kohm. Therefore the VGA cable acts like a digital to analog converter that makes the designer able to set different tones to a pixel. To be spesfici 4 bit for each color and there are three colors that determine the colorof one pixel. 2^12 is equal to 4096 colors so, 4096 different colors can be created with 12 bit RGB codes. Schematic of this module can be examined below.



(Schematic 3: Schematic of the VGA_640X480 module)

GAME_BUZZER

The buzzer module takes input as the OR gate of all buttons, so when a button is pressed a bip sound with stable note can be heard from the buzzer. This module can be enabled and disabled by usign the audio enable and disable switch. Schematic of this module can be seen below.

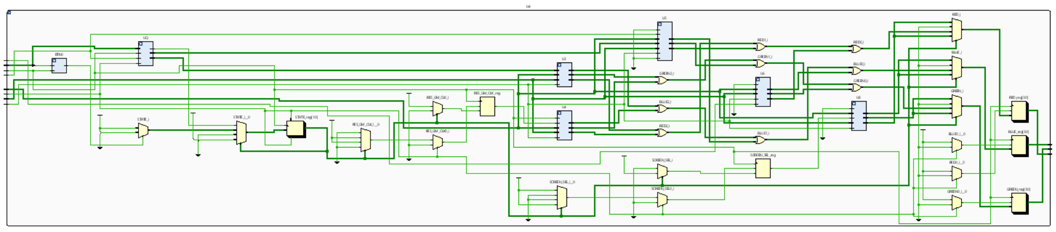


(Schematic 4: Schematic of the GAME_BUZZER module)

WHACK_A_MOLE

This module connects all modules that are used in the process of the game. A Moore type FSM with 4 states controlls the game process and changes the screen according to the different states. In the begin state the machine expects the input to be equal to '1' in order to start the game where the input is the rising edge of the top button. After the start button is pushed the machine moves to the game state where the user plays the game until game_over input become '1'. When the input become '1', the machine moves into a transition state in order to prevent glitches and moves to the end game state where the end game screen is displayed to the player.

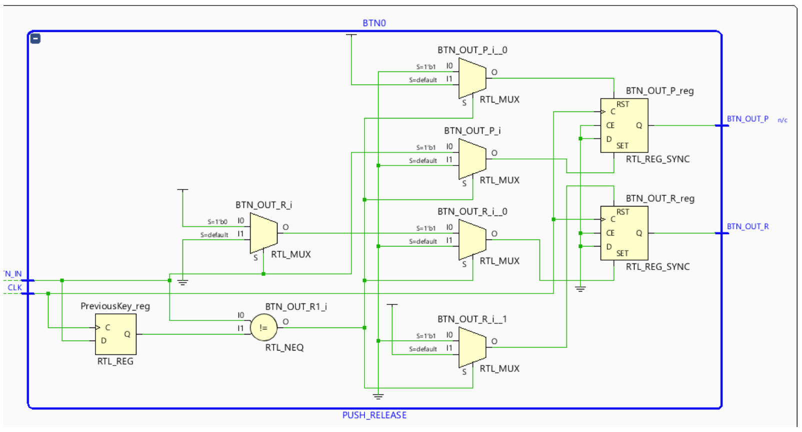
The FSM can be restarted from the begin state by using the asynchronous reset input which is controlled with one of the on-board switches. Output of the whack a mole game are R,G,B signals that will be used in the top module. Schematic of this module can be seen below.



(Schematic 5: Schematic of the WHACK_A_MOLE module)

PUSH_RELEASE

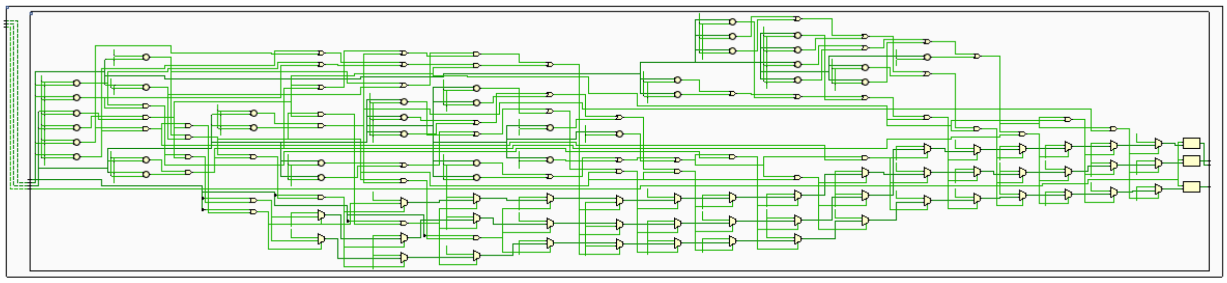
This module detects the change in the previous state of a button and current state of it. If the button value changes from '1' to '0', it indicates that the button is released and the released output become '1' ; elsif the button changes from '0' to '1', that indicates that the button is pushed so the output that indicates pushed or rising_edge of a button become '1'. This module has created to avoid repetition errors which might be caused from fast clock frequencies and long input signals caused from the player. Schematic of this module can be seen below.



(Schematic 6: Schematic of the PUSH_RELEASE module)

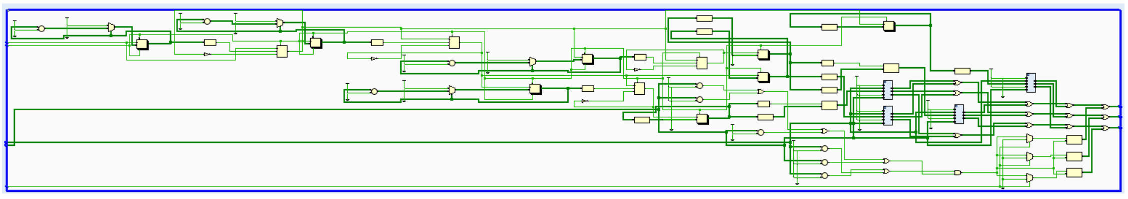
MAIN_GAME_SCREEN

This module is where I create the squares on the display and their blinking effect. The design is created by usign HC and VC signals and drawing squares by specifying margins that draw squares on the screen. The module takes input called BTN and the squares on the UI blinks according to the inputs from the BTN. Schematic of this module can be seen below.


(Schematic 7: Schematic of the MAIN_GAME_SCREEN module)

GAME_CLOCK

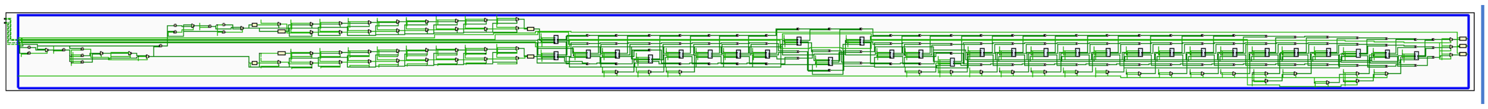
This module creates the digital clock at the bottom middle of the screen where the user can see the time that spent while playing the game. The digital clock has been created by using 4 FSM's 2 with 10 states and 2 with 6 states where they represent the transitions of seconds to decimal of seconds, that to minutes and minutes to the decimal of the minutes. The numbers used in this module has created by using the submodule char_placer which takes HC and VC as inputs and sends R,G,B signals as outputs. Schematic of this module can be seen below.


(Schematic 8: Schematic of the GAME_CLOCK module)

GAME_MODE

Game mode is the module that places the game mode indicator at the top left of the screen. In the previous plan of the project there supposed to be singleplayer and mutliplayer modes of the game but since it is cancelled this module only works for placing the single game mode in the game on the screen. This module takes HC and VC as input and place letters by usign char_placer module to the screen.

Score data is also turned into numeric notation from decimal in this module and represented at the top right corner of the screen. Score data comes from the score count module. Schematic of this module can be seen below.

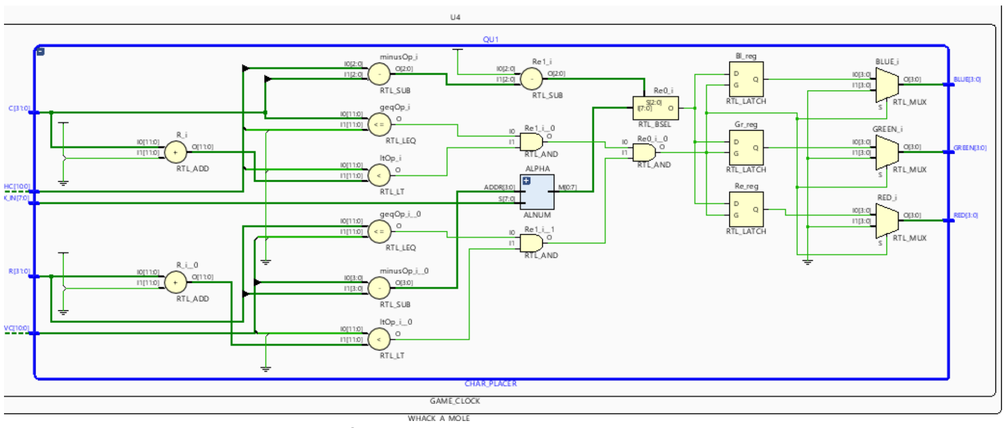


(Schematic 9: Schematic of the GAME_MODE module)

CHAR_PLACER

This module uses Alnum as a submodule and places elements in the alnum to the desired (VC,HC) point in the screen. It takes HC and VC as the input and returns R,G,B codes corresponding to that alphanumeric value on the spesified location on the screen. This module is the most used module of the project.

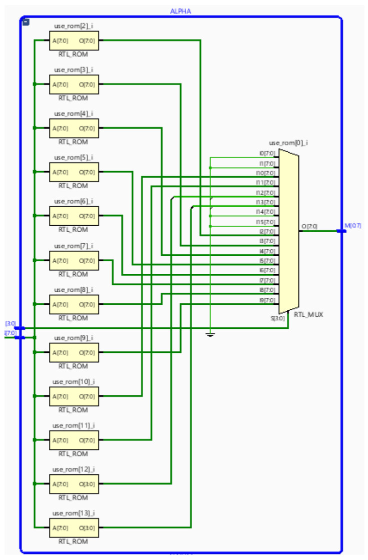
Main idea behind this moduel is to setting RGB values to all 1111 when the correct HC and VC values came from the VGA module. Those values are predetermined by selecting columns and rows of the array with a placing algorithm that I have taken help from the Digital Design book by LSE Books. Schematic of this module can be seen below.



(Schematic 10: Schematic of the CHAR_PLACER module)

ALNUM

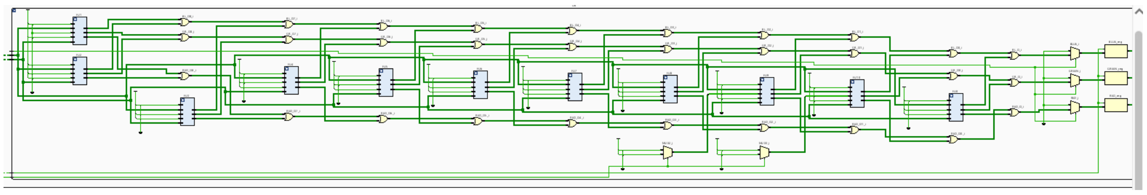
This module consists of classical alphanumeric charachters in the format of 16 byte ROM’s. A multiplexer chooses the corresponding ROM accordng to its select input and took the ROM as the output ROM. There are also additional items like up arrow, down arrow, note symbol and punctuation marks. Schematic of this module can be seen below.



(Schematic 11: Schematic of the ALNUM module)

GAME_SOUNDS

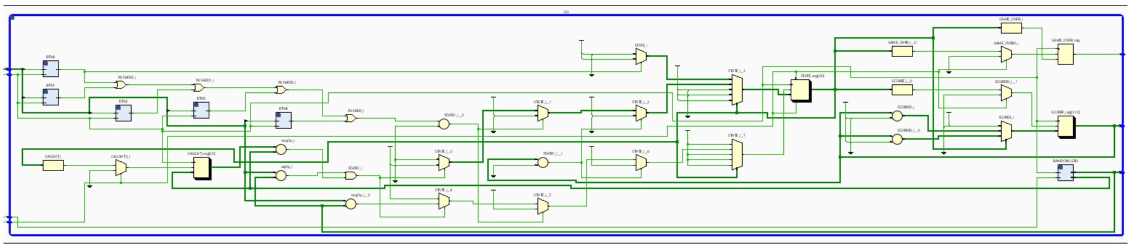
This module types AUDIO : ON or AUDIO : OFF to the bottom right corner of the screen depending on the multiplexer value which is controlled with a switch at the top module. The same switch controls the buzzer input, therefore they are all tied to each other. The module returns R,G,B as output depending on the HC and VC. Schematic of this module can be seen below.


(Schematic 12: Schematic of the GAME_SOUNDS module)

SCORE_COUNT

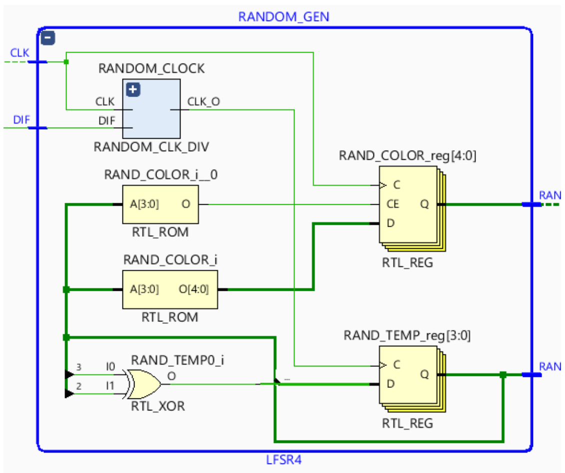
Score count module compares the input come from the buttons and the random color come from the random_generator module. If they match the FSM moves to a state that blocks score incrementing with the same square and avoids cheating. The FSM in this module is a Mealy Machine since the output is dependent on the input which are the buttons. The module sends the score and the game_over signal as the outputs.

The score count starts from 10 and increases when BTN matches with the random color appeared at that moment, and the score decreases when a wrong button is pushed. If no button is pushed to score remains the same. The game which is the FSM can be turn back to the inital state with a reset input where the score resets to 10 in the inital state. This module is connected to all other modules and feeds them with its outputs to finally creating a graphically playable game. Schematic of this module can be seen below.


(Schematic 13: Schematic of the SCORE_COUNT module)

LFSR4

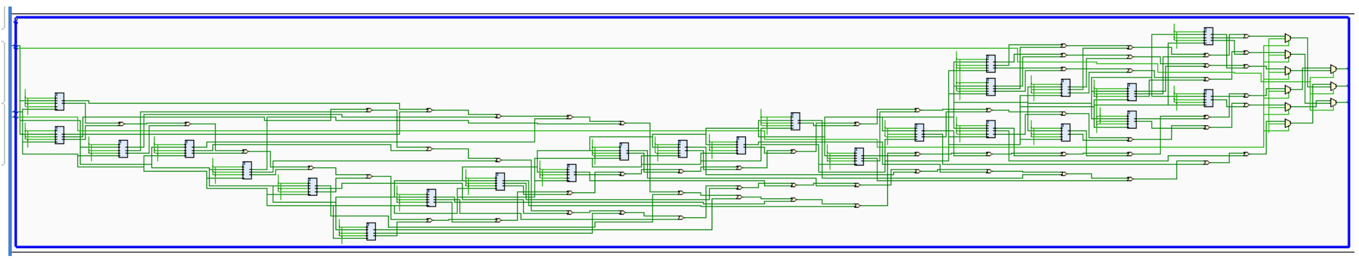
This module is written for generating random numbers by using an LFSR algorithm which consists of a 4-bit shift register with two of them has their inputs as XOR'ed outputs of other two flip flops. A one hot array which is the identical matrix of R^5 is defiend as the color vectors. So, because there are 15 shift register outputs that can be come in random times each color can be matched with 3 different states of the shift registers. The takes the random clock as the input and gives out random 5 bit vectors that correspond to squares on the screen as outputs and also 4 bit shift register outputs as the random numbers. Frequency of this module is predfined in the random_clk_div submodule and controlled with the swith to determine the difficulty of the game. The random clock outputs are directly connected to the button inputs of the main module to blink the colors in the same sequence with the random number sequence. Schematic of this module can be seen below.



(Schematic 14: Schematic of the LFSR4 module)

GAME_END_BEGIN_W

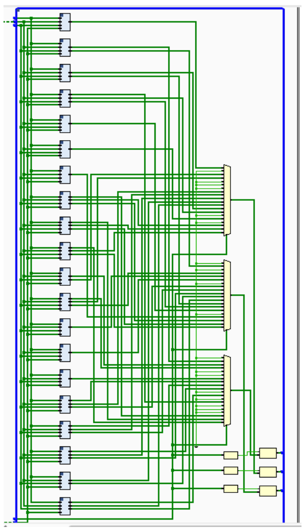
This module consists of word selector modules and a multiplexer. By connecting word_selector modules with an OR operator and multiplexing it according to if it is the game over screen or the begin screen. As a result, the module outputs R,G,B values depending on the HC and VC values. Schematic of this module can be examined below.



(Schematic 15: Schematic of the GAME_END_BEGIN_W module)

WORD_SELECTOR

Word selector module is used for multiplexing word modules which will be introduced below. Word selector module requires a multiplexer value which match with a word witch horizontal and vertical coordinates. The module returns red, green, blue values which basically can be OR'd with other R,G,B signals in order to use them on the same screen with the layer mechanism which is described above. Words in the format of modules are defined as submodules in this module and connected to a multiplexer. Schematic of this module can be seen below.

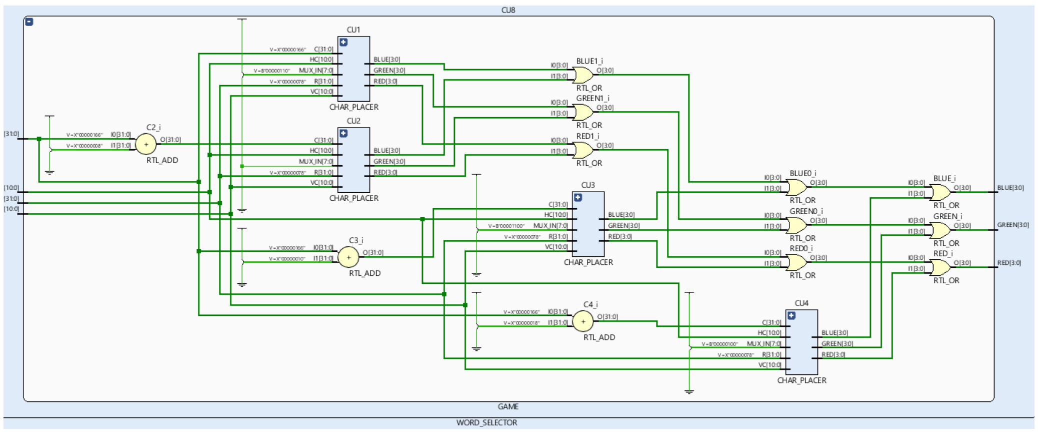


(Schematic 16:Schematic of the WORD_SELECTOR module)

CORRECT, PLAYING, MEMORY, …, W_NEXT

These modules are consist of char_placer submodules. In order to create the desired word. Each letter of the word is placed 8 bits away from each other at the same row. So their HC and VC values are like the following.

If an 4 letter word is desired to placed at the location with coordinates (VC,HC) then the algorithm is basically connecting HC of each to the same HC value and connecting their VC signals like VC , VC + 8, VC + 16, VC + 24. So, the word modules basiaclly place letters sequantially in order to create ready-to-use words. Some of the word modules are unnecessary and haven't used in the project because they were prepared for the previous design. Schematic of one of these modules can be seen below.

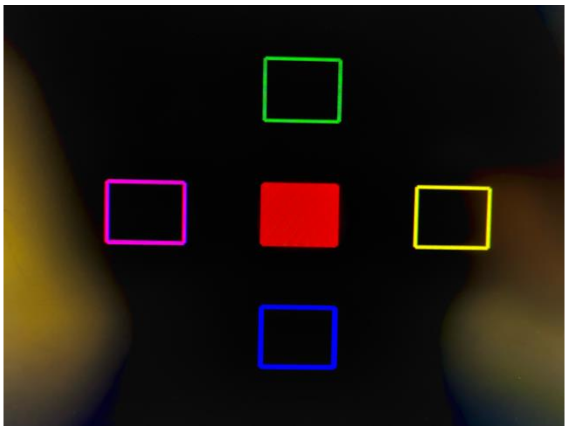


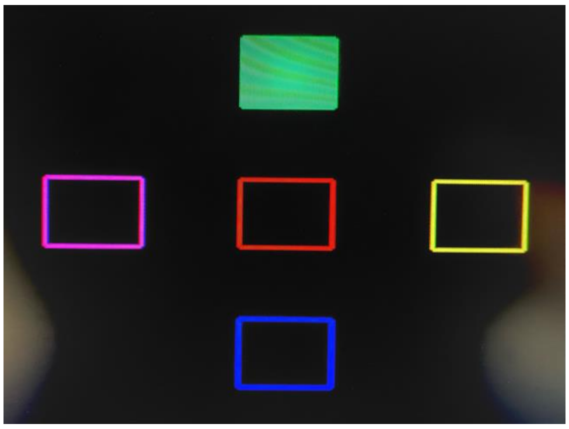
(Schematic 17: Schematic of the GAME module)

RESULTS

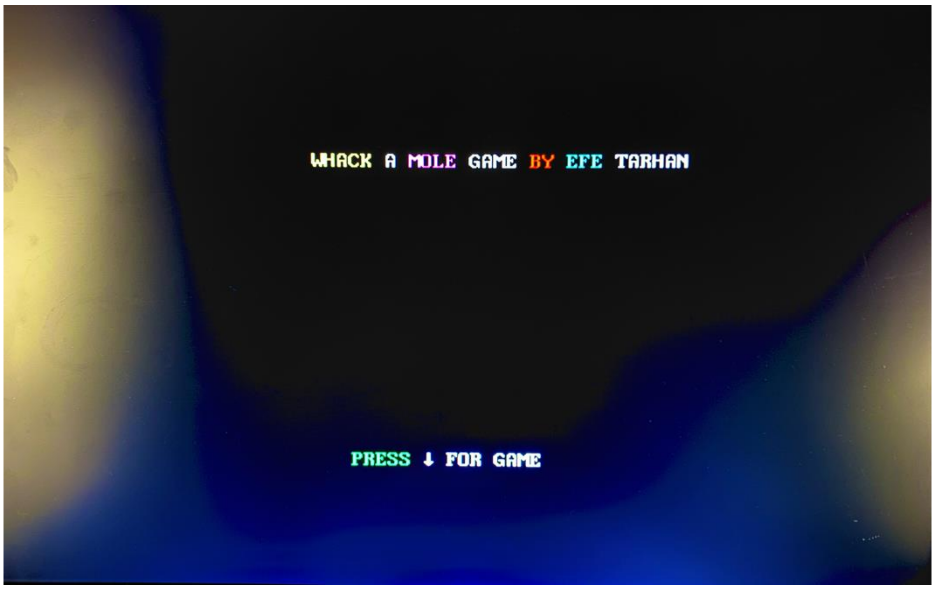
As stated above in the methodology part the game has worked properly as expected and results below are acchieved.

The main game module works properly and assigned squares blink when random numbers corresponding to them comes out from the random generator. Two different gamescreens can be seen above as the result of the properly working main game module.


(Image 1: Gamehub with 5 different colored squares)

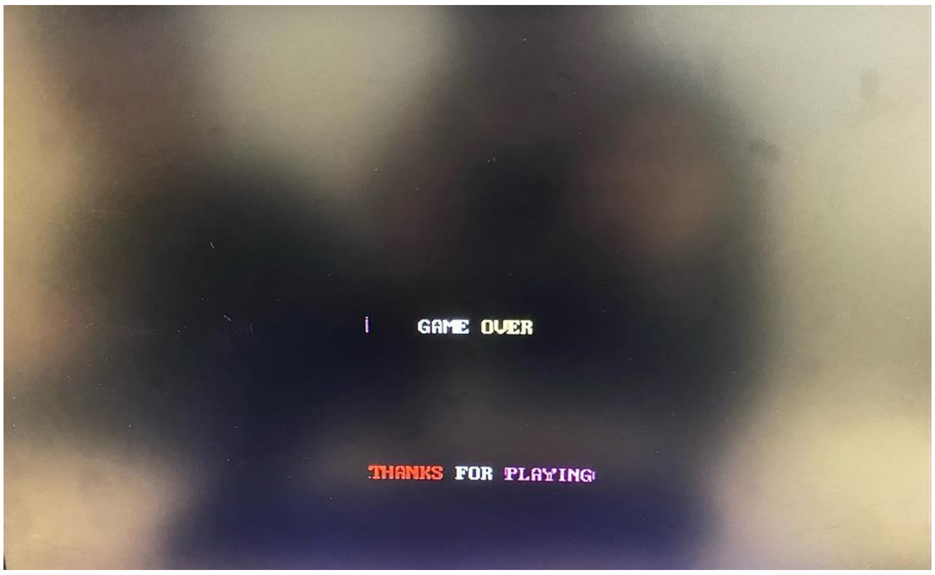

(Image 2: Gamehub with 5 different colored squares v2)

Game end begin module also worked according to the written code. Depending on the states of the game following two screens appeared on the monitor. When the up button is pressed the screen has changed from the begin screen to the main game screen and stayed in that screen until the game is over.



(Image 3: Game begin screen)

The program stays in the game over screen until the game is restarted. The game over screen can be seen below

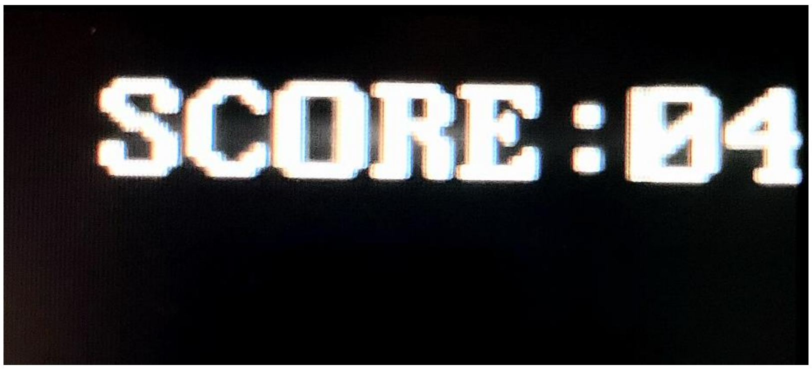


(Image 4: Game over screen)

Game difficulty indicator appeared at the bottom right corner as expected. By changing the switch game can be turned from easy mode to hard mode. This change is also indicated with words as it can be seen below.

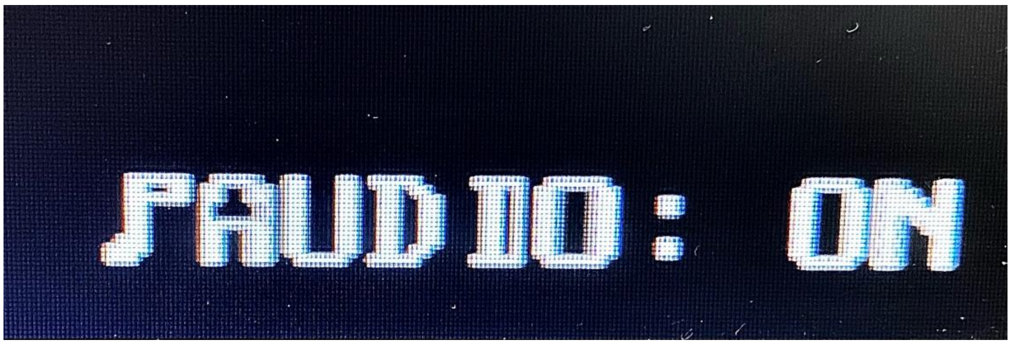
(Image 5: Difficulty level located at the bottom left corner). (Image 6: Difficulty level located at the bottom left corner v2)

Game score is also indicated at the top right corner as spesified in the methodology part. When the user pushes the correct button with correct timing the score increases and else it decreases.

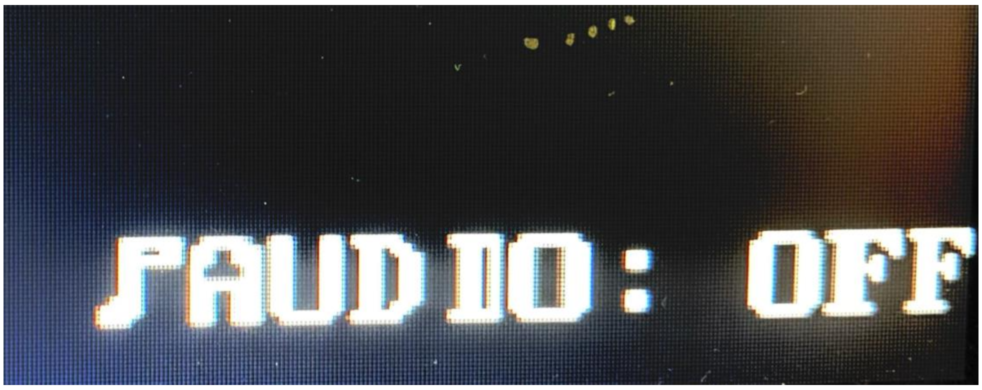


(Image 7: Score indicator located at the top right corner of the screen)

Audio on and off indicator is located at the bottom left corner of the screen. When audio is turned of with the switch it says audio off and when it is on, it says audio on.

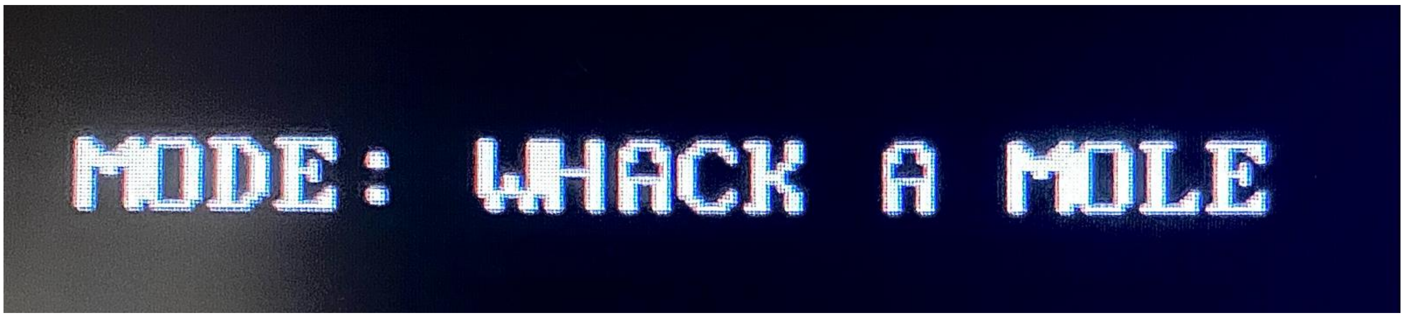


(Image 8: Audio ON indicator)



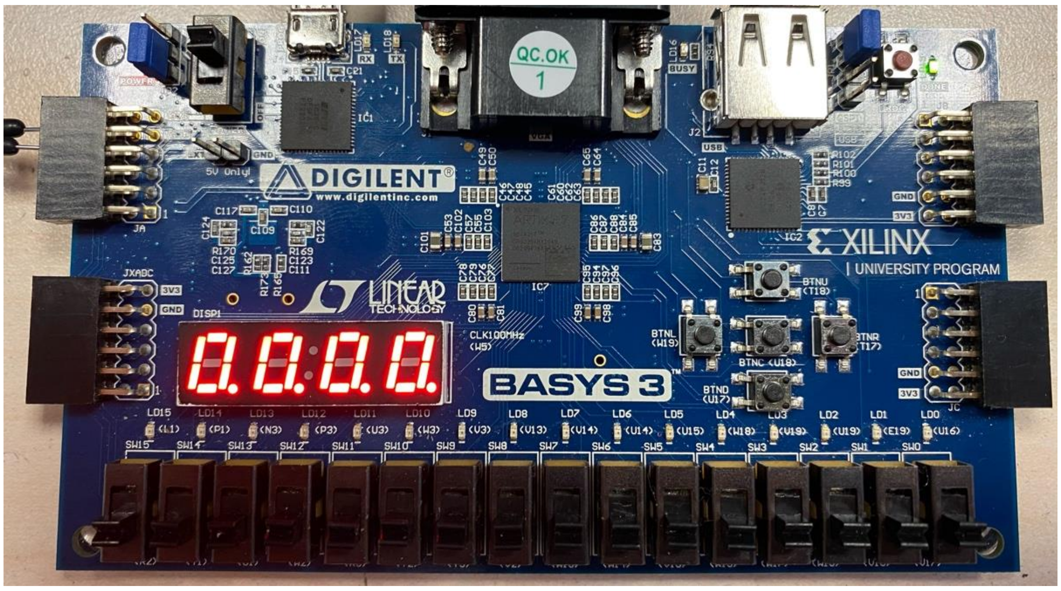
(Image 9: Audio OFF indicator)

Also, the game mode can be seen at the top left corner of the screen which is the same with the name of the game.



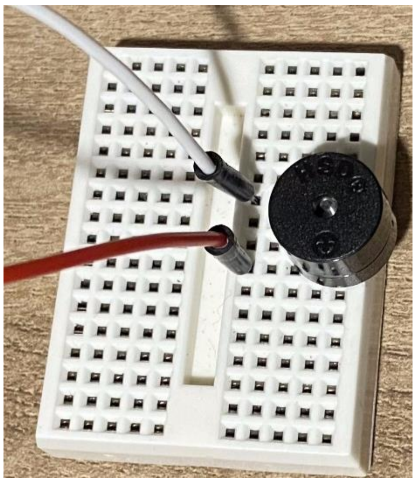
(Image 10: Mode of the game which is located at the top right corner of the screen)

Also the location of the switches that control the game are stated below with their functions.



(Image 11: Functions of the BASYS3 switches in the game : 1st switch : Screen saver, 2nd switch : difficulty mode, 15th switch : audio on/off switch, 16th switch : game reset switch)

Lastly, the buzzer which create the bip sound when a button is pressed can be seen below as the last part of the game set-up.



(Image 12: Buzzer set-up)

CONCLUSION

In this experiment, a game has game has been designed by using VGA displaying system. There were many benefits of this experiment to my understanding of VHDL and logical design.

First of all, concept of perception of eye again appeared while trying to use VHDL. I have learned how old televisions worked by understanding and implementing the theory behind showing pixels fast enough to create a whole picture in human eye. Although it is not hard to understand many problems have appeared because of this concept, for example when you assign different red, green, blue values to the I/O pins of the VGA cable according to the HC and VC an image appears on the screen ; but if you try to change the "image" on the screen faster than the frequency of the clock of VGA screen, the monitor got confuses and cannot understand what to show. Understanding the reason of this problem has taken a lot of time. But in conclusion it is understood and process clocks has set accordingly.

Second major problem occured during writing the algorithm of the game. Planned game had a slightly different algorithm than the final one but I have changed it because of the fruitless hours I have spent debugging the problems in the game algorithm, which later understood that it was connected with process timing problems and corrupted else statements. This problem has solved by changing the game algoritm because of the limited time that I have left.

Understanding and implementing an algorithm about how to place alphanumeric characters on the screen has became another problem. After several hours of thinking a solution has found which was to build a multiplexer which give out arrays that are ROM outputs and defining this module in another module which places this alphanumeric representation on the screen according to HC and VC locations. But still it was not that efficient when it come to place words on the screen so, some of the words has pre-defined as modules and again re-defined inside a multiplexer module which places the words in the desired locations. But unfortunately after changing the algorithm of the game most of these words come up to be unnecessery but since deleting a word from the project takes a lot of time, I have left them in the module but it must be known that several of them are unnecesseray and haven't used in the project.

This project also thought me the importance of another thing. I have tested each of my module with leds on the BASYS3 to understand that they work properly. I have found this way useful because finding, tracing and fixing problems in the submodules took more time from me.

As a conclusion the project made me gain many new skills not restricted with VHDL. They also contain algorithm and problem solving abilites.

YOUTUBE VIDEO LINK

https://youtu.be/Wo5Ogof4PMQ

REFERENCES

- Wikipedia contributors. (2021, November 7). *Linear-feedback shift register*. Wikipedia. https://en.wikipedia.org/wiki/Linear-feedback_shift_register?wprov=sfti1
- Haskell, R. E. (2012). *Digital Design Using Digilent FPGA Boards : VHDL/Active-HDL ed.* (2nd ed.). LBE Books.

APPENDIX

VGA_TOP

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

-----------------------------------------------

entity VGA_TOP is

PORT(

BTN : in STD_LOGIC_VECTOR(4 downto 0);

MCLK : in STD_LOGIC;

AUDIO_SW,RESTART_SW,MODE_SW : in STD_LOGIC;

VSYNC : out STD_LOGIC;

HSYNC : out STD_LOGIC;

RED : out std_logic_vector(3 downto 0);

GREEN : out std_logic_vector(3 downto 0);

BLUE : out std_logic_vector(3 downto 0);

SOUND_OUT : out std_logic;

DIF : IN STD_LOGIC

);

end VGA_TOP;

---------------------------------

architecture VGA_TOP of VGA_TOP is

---------------------------------

COMPONENT VGA_640X480

PORT(

CLK : in STD_LOGIC;

CLR : in STD_LOGIC;

VSYNC : out STD_LOGIC;

HSYNC : out STD_LOGIC;

HC : out STD_LOGIC_VECTOR (10 downto 0);

VC : out STD_LOGIC_VECTOR (10 downto 0);

VIDON : out STD_LOGIC

);

END COMPONENT;

---------------------------------

COMPONENT CLOCK_DIVIDER

PORT(

CLK_IN,RST: in std_logic;
CLOCK_OUT_HZ: in integer:=100000000;

CLK_OUT: out std_logic;

CLK_SEC_OUT: out std_logic

);

END COMPONENT;

---------------------------------

COMPONENT GAME_BUZZER

PORT(

BIP : OUT STD_LOGIC;

ON_OFF : IN STD_LOGIC;

CLICK : IN STD_LOGIC

);

END COMPONENT;

---------------------------------

COMPONENT WHACK_A_MOLE

PORT(

RESTART : in STD_LOGIC;

AUDIO_SEL : in STD_LOGIC;

VIDON : in STD_LOGIC;

CLK : in STD_LOGIC;

HC : in STD_LOGIC_VECTOR(10 downto 0);

VC : in STD_LOGIC_VECTOR(10 downto 0);

RED : out STD_LOGIC_VECTOR(3 downto 0);

GREEN : out STD_LOGIC_VECTOR(3 downto 0);

BLUE : out STD_LOGIC_VECTOR(3 downto 0);

BTN : in STD_LOGIC_VECTOR(4 DOWNTO 0);

DIF : IN STD_LOGIC

);

END COMPONENT;

---------------------------------

SIGNAL CLK25,CLK1,CLR,VIDON,CLICK: STD_LOGIC;

SIGNAL HC,VC : STD_LOGIC_VECTOR(10 DOWNTO 0);

SIGNAL R1,G1,B1,R2,G2,B2 : STD_LOGIC_VECTOR(3 DOWNTO 0);

--------------------------------

BEGIN

--------------------------------

U1: CLOCK_DIVIDER
```

```
PORT MAP(
CLK_IN => MCLK,
RST => CLR,
CLOCK_OUT_HZ => 25000000,
CLK_OUT => CLK25,
CLK_SEC_OUT => CLK1
);
--------------------------------
U2: VGA_640X480
PORT MAP(
CLK => CLK25,
CLR => CLR,
HSYNC => HSYNC,
VSYNC => VSYNC,
HC => HC,
VC => VC,
VIDON => VIDON
);
--------------------------------
U3 : GAME_BUZZER
PORT MAP(
BIP => SOUND_OUT,
ON_OFF => AUDIO_SW,
CLICK => CLICK
);
-----------------------------------
U4 : WHACK_A_MOLE
PORT MAP(
RESTART => RESTART_SW,
AUDIO_SEL => AUDIO_SW,
VIDON => VIDON,
CLK => MCLK,
HC => HC,
VC => VC,
RED => R1,
GREEN => G1,
BLUE => B1,
BTN => BTN,
DIF => DIF
);

CLICK <= BTN(0) OR BTN(1) OR BTN(2) OR BTN(3) OR BTN(4);

PROCESS(MODE_SW)
BEGIN
CASE MODE_SW IS
--SCREEN_ON
WHEN '0' => RED <= R1; GREEN <= G1; BLUE <= B1;
--BLANK SCREEN
WHEN '1' => RED <= X"0"; GREEN <= X"0"; BLUE <= X"0";
END CASE;
END PROCESS;

end VGA_TOP;
```

CLOCK_DIVIDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity CLOCK_DIVIDER is
port (
CLK_IN,RST: in std_logic;
CLOCK_OUT_HZ: in integer:=50000000;
CLK_OUT: out std_logic;
CLK_SEC_OUT : out std_logic);
end CLOCK_DIVIDER;

architecture CLOCK_DIVIDER of CLOCK_DIVIDER is

signal COUNT: integer:=1;
signal COUNT2 : integer := 1;
signal TMP : std_logic := '0';
signal TMP2 : std_logic := '0';
```

```
begin

process(TMP,CLK_IN,RST)
begin
if(RST='1') then
COUNT <= 1;
TMP <= '0';
elsif(rising_edge(CLK_IN)) then
count <=count+1;
if (count = 50000000/clock_out_hz) then
TMP <= NOT TMP;
count <= 1;
end if;
end if;
CLK_OUT <= TMP;
end process;
process(TMP2,CLK_IN,RST)
begin
if(RST='1') then
COUNT2 <= 1;
TMP2 <= '0';
elsif(rising_edge(CLK_IN)) then
COUNT2 <= COUNT2 + 1;
if (COUNT2 = 50000000) then
TMP2 <= NOT TMP2;
COUNT2 <= 1;
end if;
end if;
CLK_SEC_OUT <= TMP2;
end process;
end CLOCK_DIVIDER;
```

VGA_640X480

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity VGA_640X480 is
port(
CLR : in std_logic;
CLK : in std_logic;
HSYNC : out std_logic;
VSYNC : out std_logic;
HC : out std_logic_vector(10 downto 0);
VC : out std_logic_vector(10 downto 0);
VIDON : out std_logic
);
end VGA_640X480;

architecture VGA_640X480 of VGA_640X480 is

-- maximum value for the horizontal pixel counter
constant HMAX  : std_logic_vector(10 downto 0) := "01100100000"; -- 800
-- maximum value for the vertical pixel counter
constant VMAX  : std_logic_vector(10 downto 0) := "01000001101"; -- 525
-- total number of visible columns
constant HLINES: std_logic_vector(10 downto 0) := "01010000000"; -- 640
-- value for the horizontal counter where front porch ends
constant HFP   : std_logic_vector(10 downto 0) := "01010001000"; -- 648
-- value for the horizontal counter where the synch pulse ends
constant HSP   : std_logic_vector(10 downto 0) := "01011101000"; -- 744
-- total number of visible lines
constant VLINES: std_logic_vector(10 downto 0) := "00111100000"; -- 480
-- value for the vertical counter where the front porch ends
constant VFP   : std_logic_vector(10 downto 0) := "00111100010"; -- 482
-- value for the vertical counter where the synch pulse ends
constant VSP   : std_logic_vector(10 downto 0) := "00111100100"; -- 484
-- polarity of the horizontal and vertical synch pulse
-- only one polarity used, because for this resolution they coincide.
constant SPP   : std_logic := '0';
```

```
-- horizontal and vertical counters
signal hcounter : std_logic_vector(10 downto 0) := (others => '0');
signal vcounter : std_logic_vector(10 downto 0) := (others => '0');

-- active when inside visible screen area.
signal video_enable: std_logic;

begin

  HC <= hcounter;
  VC <= vcounter;

  VIDON <=  video_enable when rising_edge(CLK);

  h_count: process(CLK)
  begin
    if(rising_edge(CLK)) then
      if(CLR = '1') then
        hcounter <= (others => '0');
      elsif(hcounter = HMAX) then
        hcounter <= (others => '0');
      else
        hcounter <= hcounter + 1;
      end if;
    end if;
  end process h_count;

  v_count: process(CLK)
  begin
    if(rising_edge(CLK)) then
      if(CLR = '1') then
        vcounter <= (others => '0');
      elsif(hcounter = HMAX) then
        if(vcounter = VMAX) then
          vcounter <= (others => '0');
        else
          vcounter <= vcounter + 1;
```

```
         end if;
      end if;
   end if;
 end process v_count;



 do_hs: process(CLK)
 begin
   if(rising_edge(CLK)) then
     if(hcounter >= HFP and hcounter < HSP) then
       HSYNC <= SPP;
     else
       HSYNC <= not SPP;
     end if;
   end if;
 end process do_hs;

 do_vs: process(CLK)
 begin
   if(rising_edge(CLK)) then
     if(vcounter >= VFP and vcounter < VSP) then
       VSYNC <= SPP;
     else
       VSYNC <= not SPP;
     end if;
   end if;
 end process do_vs;



 video_enable <= '1' when (hcounter < HLINES and vcounter < VLINES) else '0';

end VGA_640X480;
```

GAME_BUZZER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity GAME_BUZZER is
```

```
PORT(
BIP : OUT STD_LOGIC;
ON_OFF : IN STD_LOGIC;
CLICK : IN STD_LOGIC
);
end GAME_BUZZER;

architecture GAME_BUZZER of GAME_BUZZER is
begin

PROCESS(ON_OFF,CLICK)
BEGIN
IF ON_OFF = '1' THEN
IF CLICK = '1' THEN
BIP <= '1';
ELSE
BIP <= '0';
END IF;
ELSE
BIP <= '0';
END IF;
END PROCESS;

end GAME_BUZZER;
```

WHACK_A_MOLE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

entity WHACK_A_MOLE is
PORT(
RESTART : in STD_LOGIC;
AUDIO_SEL : in STD_LOGIC;
VIDON : in STD_LOGIC;
CLK : in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0);
BTN : in STD_LOGIC_VECTOR(4 DOWNTO 0);
DIF : IN STD_LOGIC
);
end WHACK_A_MOLE;

architecture Behavioral of WHACK_A_MOLE is


----------------------------
COMPONENT MAIN_GAME_SCREEN
PORT(
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0);
BTN : in STD_LOGIC_VECTOR(4 downto 0)
);
END COMPONENT;
----------------------------
COMPONENT GAME_END_BEGIN_W
PORT(
SEL_END_BEGIN : in STD_LOGIC;
CLR : in STD_LOGIC;
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
END COMPONENT;
-------------------------------------
COMPONENT GAME_CLOCK
```

```
PORT(
CLK : in STD_LOGIC;
CLR : in STD_LOGIC;
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
END COMPONENT;
-------------------------------------
COMPONENT GAME_MODE
PORT(
DIFFICULTY : IN STD_LOGIC;
LVL: IN integer := 0;
mode_sel : in STD_LOGIC;
CLR : in STD_LOGIC;
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
END COMPONENT;
-------------------------------------
COMPONENT GAME_SOUNDS
PORT(
audio_sel : in STD_LOGIC;
CLR : in STD_LOGIC;
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
END COMPONENT;
-------------------------------------
COMPONENT WORD_SELECTOR
PORT(
SEL_WORD : in std_logic_vector(7 downto 0);
R : in integer:= 0;
C : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;

------------------------------------------
COMPONENT SCORE_COUNT
PORT(
DIF : IN STD_LOGIC;
CLK : IN STD_LOGIC;
RESET : IN STD_LOGIC;
BTN : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
GAME_OVER : OUT STD_LOGIC;
SQUARES : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
SCORE : OUT INTEGER:= 0
);
END COMPONENT;
----------------------
COMPONENT PUSH_RELEASE IS
PORT(
CLK : IN STD_LOGIC;
BTN_IN : IN STD_LOGIC;
BTN_OUT_P : OUT STD_LOGIC;
BTN_OUT_R : OUT STD_LOGIC
);
END COMPONENT;
```

```
--------------------------
TYPE STATE_TYPE IS (S_GAME,S_OVER,S_BEGIN,S_BLANK);
SIGNAL STATE : STATE_TYPE;
SIGNAL RED1,RED2,RED3,RED4,RED5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL GREEN1,GREEN2,GREEN3,GREEN4,GREEN5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL BLUE1,BLUE2,BLUE3,BLUE4,BLUE5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL SQUARES : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL SCORE : INTEGER;
SIGNAL CLR : STD_LOGIC := '0';
SIGNAL SCREEN_SEL : STD_LOGIC;
SIGNAL GAME_OVER : STD_LOGIC;
SIGNAL RES_GM_CLK: STD_LOGIC;
SIGNAL BTNR,BTNP : STD_LOGIC;
BEGIN
-----------------------------------------
BTN0 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(2),BTN_OUT_P => BTNP,BTN_OUT_R => BTNR );
------------------------------------------
U3: MAIN_GAME_SCREEN
PORT MAP(
VIDON => VIDON,
HC => HC,
VC => VC,
RED => RED1,
GREEN => GREEN1,
BLUE => BLUE1,
BTN => SQUARES
);
-------------------------------------
U4: GAME_CLOCK
PORT MAP(
CLK => CLK,
CLR => RES_GM_CLK,
VIDON => VIDON,
HC => HC,
VC => VC,
RED => RED2,
GREEN => GREEN2,
BLUE => BLUE2
);
--------------------------------------
U5: GAME_MODE
PORT MAP(
DIFFICULTY => DIF,
LVL => SCORE,
MODE_SEL => '0',
CLR => CLR,
VIDON => VIDON,
HC => HC,
VC => VC,
RED => RED3,
GREEN => GREEN3,
BLUE => BLUE3
);
--------------------------------------
U6: GAME_SOUNDS
PORT MAP(
audio_sel => AUDIO_SEL,
CLR => CLR,
VIDON => VIDON,
HC => HC,
VC => VC,
RED => RED4,
GREEN => GREEN4,
BLUE => BLUE4
);
--------------------------------------
U8 : GAME_END_BEGIN_W
PORT MAP(
SEL_END_BEGIN => SCREEN_SEL,
CLR => CLR ,
VIDON=> VIDON,
HC => HC,
VC=> VC,
RED => RED5,
GREEN => GREEN5,
BLUE => BLUE5
);

---------------------------------------
UQ : SCORE_COUNT
PORT MAP(
DIF => DIF,
CLK => CLK,
RESET => RESTART,
BTN => BTN,
GAME_OVER => GAME_OVER,
SQUARES => SQUARES,
SCORE => SCORE
);
---------------------------------------
PROCESS(CLK)
BEGIN
IF RESTART = '1' THEN
STATE <= S_BEGIN;
ELSIF RISING_EDGE(CLK) THEN
CASE STATE IS
-----------------------------
WHEN S_BEGIN =>
SCREEN_SEL <= '0';
RED <= RED5;
GREEN <= GREEN5;
BLUE <= BLUE5;
IF BTNR = '1' THEN
STATE <= S_BLANK;
ELSE
STATE <= S_BEGIN;
END IF;

WHEN S_BLANK =>
RED <= "0000";
GREEN <= "0000";
```

```
BLUE <= "0000";
STATE <= S_GAME;
RES_GM_CLK <= '1';

WHEN S_GAME =>
RES_GM_CLK <= '0';
RED <= RED1 OR RED2 OR RED3 OR RED4;
GREEN <= GREEN1 OR GREEN2 OR GREEN3 OR GREEN4;
BLUE <= BLUE1 OR BLUE2 OR BLUE3 OR BLUE4;
IF GAME_OVER = '1' THEN
STATE <= S_OVER;
ELSE
STATE <= S_GAME;
END IF;
-----------------------------
WHEN S_OVER =>
SCREEN_SEL <= '1';
RED <= RED5;
GREEN <= GREEN5;
BLUE <= BLUE5;
STATE <= S_OVER;
END CASE;
END IF;
END PROCESS;
end Behavioral;
```

PUSH_RELEASE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PUSH_RELEASE is
PORT(
CLK : IN STD_LOGIC;
BTN_IN : IN STD_LOGIC;
BTN_OUT_P : OUT STD_LOGIC;
BTN_OUT_R : OUT STD_LOGIC
);
end PUSH_RELEASE;
```

```
architecture PUSH_RELEASE of PUSH_RELEASE is



signal PreviousKey: std_logic := '0';

signal CurrentKey: std_logic;


begin

CurrentKey <= BTN_IN;

process (clk)

begin

IF RISING_EDGE(CLK) THEN

IF (CurrentKey /= PreviousKey) THEN

      IF (CurrentKey = '0') THEN

        BTN_OUT_R <= '1';

      else

        BTN_OUT_P <= '1';

      end if;

ELSE

BTN_OUT_R <= '0';

BTN_OUT_P <= '0';

    end if;

    PreviousKey <= CurrentKey;

  end if;

end process;


end PUSH_RELEASE;
```

MAIN_GAME_SCREEN

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;


entity MAIN_GAME_SCREEN is

PORT(

VIDON: in STD_LOGIC;

HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0);
BTN : in STD_LOGIC_VECTOR(4 downto 0)
);
end MAIN_GAME_SCREEN;

architecture MAIN_GAME_SCREEN of MAIN_GAME_SCREEN is

begin

DRAW : PROCESS(BTN,HC,VC,VIDON)
BEGIN
IF (VIDON = '1') THEN

------------------------------------------------------------------------
IF (HC = 300 OR HC = 301 OR HC = 339 OR HC = 340) AND ((VC > 140 AND VC < 180)) THEN
RED <= "0000";
GREEN <= "1111";
BLUE <= "0000";
ELSIF (HC = 300 OR HC = 301 OR HC = 339 OR HC = 340) AND ((VC > 220 AND VC < 260)) THEN
RED <= "1111";
GREEN <= "0000";
BLUE <= "0000";
ELSIF (HC = 300 OR HC = 301 OR HC = 339 OR HC = 340) AND ((VC > 300 AND VC < 340)) THEN
RED <= "0000";
GREEN <= "0000";
BLUE <= "1111";
ELSIF (HC = 220 OR HC = 260 OR HC = 221 OR HC = 259) AND ((VC > 220 AND VC < 260)) THEN
RED <= "1111";
GREEN <= "0000";
BLUE <= "1111";
ELSIF (HC = 380 OR HC = 420 OR HC = 381 OR HC = 419) AND ((VC > 220 AND VC < 260)) THEN
RED <= "1111";
GREEN <= "1111";
BLUE <= "0000";
```

```
--------------------------------------------------------------------------------
ELSIF (VC = 220 OR VC = 260 OR VC = 221 OR VC = 259) AND ((HC > 220 AND HC < 260)) THEN
RED <= "1111";
GREEN <= "0000";
BLUE <= "1111";
ELSIF (VC = 220 OR VC = 260 OR VC = 221 OR VC = 259) AND ((HC > 300 AND HC < 340)) THEN
RED <= "1111";
GREEN <= "0000";
BLUE <= "0000";
ELSIF (VC = 220 OR VC = 260 OR VC = 221 OR VC = 259) AND ((HC > 380 AND HC < 420)) THEN
RED <= "1111";
GREEN <= "1111";
BLUE <= "0000";
ELSIF (VC = 140 OR VC = 180 OR VC = 141 OR VC = 179) AND ((HC > 300 AND HC < 340)) THEN
RED <= "0000";
GREEN <= "1111";
BLUE <= "0000";
ELSIF (VC = 300 OR VC = 340 OR VC = 301 OR VC = 339) AND ((HC > 300 AND HC < 340)) THEN
RED <= "0000";
GREEN <= "0000";
BLUE <= "1111";
--------------------------------------------------------------------------------
ELSIF (((HC > 300 OR HC < 340) AND ((VC > 220 AND VC < 260) OR (VC > 300 AND VC < 340) OR (VC < 180 AND VC
> 140))) OR
(((HC < 260 OR HC > 220) OR ( HC > 380 OR HC < 420)) AND (VC > 220 AND VC < 260))) THEN


RED <= "0000";
GREEN <= "0000";
BLUE <= "0000";


  IF (BTN(0) = '1') AND ((HC > 300 AND HC < 340 ) AND(VC < 180 AND VC > 140 )) THEN
  RED <= "0000";
  GREEN <= "1111";
  BLUE <= "0000";
  ELSIF (BTN(1) = '1') AND ((HC > 300 AND HC < 340 ) AND(VC < 260 AND VC > 220 )) THEN
  RED <= "1111";
  GREEN <= "0000";
```

```
   BLUE <= "0000";
   ELSIF (BTN(2) = '1') AND ((HC > 300 AND HC < 340 ) AND(VC < 340 AND VC > 300 )) THEN
   RED <= "0000";
   GREEN <= "0000";
   BLUE <= "1111";
   ELSIF (BTN(3) = '1') AND ((HC < 260 AND HC > 220 ) AND(VC < 260 AND VC > 220 )) THEN
   RED <= "1111";
   GREEN <= "0000";
   BLUE <= "1111";
   ELSIF (BTN(4) = '1') AND ((HC < 420 AND HC > 380 ) AND (VC < 260 AND VC > 220 )) THEN
   RED <= "1111";
   GREEN <= "1111";
   BLUE <= "0000";

END IF;
-----------------------------------------------------------------------------------------------------------------------
ELSE
RED <= "0000";
GREEN <= "0000";
BLUE <= "0000";

END IF;
END IF;



END PROCESS;
end MAIN_GAME_SCREEN;
```

GAME_CLOCK

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity GAME_CLOCK is
PORT(
CLK : in STD_LOGIC;
CLR : in STD_LOGIC;
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
end GAME_CLOCK;
architecture GAME_CLOCK of GAME_CLOCK is
type state_type is(STATE0,STATE1,STATE2,STATE3,STATE4,STATE5,STATE6,STATE7,STATE8,STATE9);
signal P1_STATE,P2_STATE,P3_STATE,P4_STATE ,N1_STATE,N2_STATE,N3_STATE,N4_STATE : state_type;
---------------------------------------------------------------------------
signal sec_clock, sec_dec_clock,min_clock,min_dec_clock : STD_LOGIC;
signal sec_clock_count,min_clock_count :integer := 1;
signal min_dec_clock_count,sec_dec_clock_count:integer := -4;
signal sec_c,sec_d_c,min_c,min_d_c : integer := 0;
--------------------------------------------------------
signal TMP1,TMP2,TMP3,TMP4 : STD_LOGIC;
signal MUX1,MUX2,MUX3,MUX4 : STD_LOGIC_VECTOR(7 downto 0);
-------------------------------------------------------------------
signal RED1,RED2,RED3,RED4,RED5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
signal BLUE1,BLUE2,BLUE3,BLUE4,BLUE5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
signal GREEN1,GREEN2,GREEN3,GREEN4,GREEN5 : STD_LOGIC_VECTOR(3 DOWNTO 0);

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;


--------------------------------------------------------------------------------
```

```
begin
-----------------------------------------------------------------------------
QU1 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX1,
R => 403,
C => 300,
HC => HC,
VC => VC,
RED => RED1,
GREEN => GREEN1,
BLUE => BLUE1
);
--------------------
QU2 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX2,
R => 403,
C => 308,
HC => HC,
VC => VC,
RED => RED2,
GREEN => GREEN2,
BLUE => BLUE2
);
----------------------
QU3 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX3,
R => 403,
C => 324,
HC => HC,
VC => VC,
RED => RED3,
GREEN => GREEN3,
BLUE => BLUE3
);
```

```
---------------------
QU4 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX4,
R => 403,
C => 332,
HC => HC,
VC => VC,
RED => RED4,
GREEN => GREEN4,
BLUE => BLUE4
);

--------------------------------- PROCESS SEC_CLOCK--------------------------------
process(TMP1,CLK,CLR)
begin
if(CLR = '1') then
sec_clock_count <= 1;
TMP1 <= '0';
elsif(rising_edge(CLK)) then
sec_clock_count <= sec_clock_count + 1;
if (sec_clock_count = 50000000) then
TMP1 <= NOT TMP1;
sec_clock_count <= 1;
end if;
end if;
sec_clock <= TMP1;
end process;

-------------------------------------SEC_DEC_CLOCK---------------------------------
process(TMP2,sec_clock,CLR)
begin
if(CLR = '1') then
sec_dec_clock_count <= -4;
TMP2 <= '0';
elsif(rising_edge(sec_clock)) then
sec_dec_clock_count <= sec_dec_clock_count + 1;
```

```
if (sec_dec_clock_count = 5) then
TMP2 <= NOT TMP2;
sec_dec_clock_count <= 1;
end if;
end if;
sec_dec_clock <= TMP2;
end process;
-----------------------------------------------MIN_CLOCK------------------------------
process(sec_clock,CLR,TMP3)
begin
if(CLR = '1') then
min_clock_count <= 1;
TMP3 <= '0';
elsif(rising_edge(sec_clock)) then
min_clock_count <= min_clock_count + 1;
if (min_clock_count = 60) then
TMP3 <= NOT TMP3;
min_clock_count <= 1;
end if;
end if;
min_clock <= TMP3;
end process;
-------------------------------------------------MIN_DEC_CLOCK--------------------------
process(min_clock,CLR,TMP4)
begin
if(CLR = '1') then
min_dec_clock_count <= -4;
TMP4 <= '0';
elsif(rising_edge(min_clock)) then
min_dec_clock_count <= min_dec_clock_count + 1;
if (min_dec_clock_count = 5) then
TMP4 <= NOT TMP4;
min_dec_clock_count <= 1;
end if;
end if;
min_dec_clock <= TMP4;
end process;
```

```
---------------------------------------SEC_STATE_MACHINE--------------------------------------
S1REG : PROCESS(CLR,sec_clock)
BEGIN
IF CLR  = '1' THEN
P1_STATE <= STATE0;
ELSIF RISING_EDGE(sec_clock) THEN
P1_STATE <= N1_STATE;
END IF;
END PROCESS;
---------------------------------------SEC_DEC_STATE_MACHINE--------------------------------------
S2REG : PROCESS(sec_dec_clock,CLR)
BEGIN
IF CLR  = '1' THEN
P2_STATE <= STATE0;
ELSIF RISING_EDGE(sec_dec_clock) THEN
P2_STATE <= N2_STATE;
END IF;
END PROCESS;
---------------------------------------MIN_STATE_MACHINE--------------------------------------
S3REG : PROCESS(min_clock,CLR)
BEGIN
IF CLR  = '1' THEN
P3_STATE <= STATE0;
ELSIF RISING_EDGE(min_clock) THEN
P3_STATE <= N3_STATE;
END IF;
END PROCESS;
---------------------------------------MIN_DEC_STATE_MACHINE--------------------------------------
S4REG : PROCESS(min_dec_clock,CLR)
BEGIN
IF CLR  = '1' THEN
P4_STATE <= STATE0;
ELSIF RISING_EDGE(min_dec_clock) THEN
P4_STATE <= N4_STATE;
END IF;
END PROCESS;
----------------------------------------------STATE MACHINES AND SCREEN_DISPLAYING-----------------
```

```
C1 : PROCESS(P1_STATE)
BEGIN
CASE P1_STATE IS
WHEN STATE0 =>
MUX4 <= X"24";
N1_STATE <= STATE1;
WHEN STATE1 =>
MUX4 <= X"1A";
N1_STATE <= STATE2;
WHEN STATE2 =>
MUX4 <= X"1B";
N1_STATE <= STATE3;
WHEN STATE3 =>
MUX4 <= X"1C";
N1_STATE <= STATE4;
WHEN STATE4 =>
MUX4 <= X"1D";
N1_STATE <= STATE5;
WHEN STATE5 =>
MUX4 <= X"1E";
N1_STATE <= STATE6;
WHEN STATE6 =>
MUX4 <= X"1F";
N1_STATE <= STATE7;
WHEN STATE7 =>
MUX4 <= X"20";
N1_STATE <= STATE8;
WHEN STATE8 =>
MUX4 <= X"21";
N1_STATE <= STATE9;
WHEN STATE9 =>
MUX4 <= X"22";
N1_STATE <= STATE0;
END CASE;
END PROCESS;

C2 : PROCESS(P2_STATE)
```

```
BEGIN
CASE P2_STATE IS
WHEN STATE0 =>
MUX3 <= X"24";
N2_STATE <= STATE1;
WHEN STATE1 =>
MUX3 <= X"1A";
N2_STATE <= STATE2;
WHEN STATE2 =>
MUX3 <= X"1B";
N2_STATE <= STATE3;
WHEN STATE3 =>
MUX3 <= X"1C";
N2_STATE <= STATE4;
WHEN STATE4 =>
MUX3 <= X"1D";
N2_STATE <= STATE5;
WHEN STATE5 =>
MUX3 <= X"1E";
N2_STATE <= STATE0;
WHEN OTHERS =>
N2_STATE <= STATE0;
END CASE;
END PROCESS;

C3 : PROCESS(P3_STATE)
BEGIN
CASE P3_STATE IS
WHEN STATE0 =>
MUX2 <= X"24";
N3_STATE <= STATE1;
WHEN STATE1 =>
MUX2 <= X"1A";
N3_STATE <= STATE2;
WHEN STATE2 =>
MUX2 <= X"1B";
N3_STATE <= STATE3;
```

```
WHEN STATE3 =>

MUX2 <= X"1C";

N3_STATE <= STATE4;

WHEN STATE4 =>

MUX2 <= X"1D";

N3_STATE <= STATE5;

WHEN STATE5 =>

MUX2 <= X"1E";

N3_STATE <= STATE6;

WHEN STATE6 =>

MUX2 <= X"1F";

N3_STATE <= STATE7;

WHEN STATE7 =>

MUX2 <= X"20";

N3_STATE <= STATE8;

WHEN STATE8 =>

MUX2 <= X"21";

N3_STATE <= STATE9;

WHEN STATE9 =>

MUX2 <= X"22";

N3_STATE <= STATE0;

END CASE;

END PROCESS;



C4 : PROCESS(P4_STATE)

BEGIN

CASE P4_STATE IS

WHEN STATE0 =>

MUX1 <= X"24";

N4_STATE <= STATE1;

WHEN STATE1 =>

MUX1 <= X"1A";

N4_STATE <= STATE2;

WHEN STATE2 =>

MUX1 <= X"1B";

N4_STATE <= STATE3;
```

```
WHEN STATE3 =>

MUX1 <= X"1C";

N4_STATE <= STATE4;

WHEN STATE4 =>

MUX1 <= X"1D";

N4_STATE <= STATE5;

WHEN STATE5 =>

MUX1 <= X"1E";

N4_STATE <= STATE0;

WHEN OTHERS =>

N4_STATE <= STATE0;

END CASE;

END PROCESS;

-----------------------------------------------------------CLOCK_ELEMENTS-----------------------------------------

PROCESS(HC,VC,VIDON)

BEGIN

IF VIDON = '1' THEN

IF (HC = 319 OR HC = 320) AND (VC = 407 OR VC = 408 OR VC= 412 OR VC= 413 ) THEN

RED5 <= "1111";

GREEN5 <= "1111";

BLUE5 <= "1111";

END IF;

END IF;

END PROCESS;


RED <= RED1 OR RED2 OR RED3 OR RED4 OR RED5;

GREEN <= GREEN1 OR GREEN2 OR GREEN3 OR GREEN4 OR GREEN5;

BLUE <= BLUE1 OR BLUE2 OR BLUE3 OR BLUE4 OR BLUE5;


end GAME_CLOCK;
```

GAME_MODE

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

----------------------------

-- 0 => WHACK A MOLE

---------------------------
```

```
entity GAME_MODE is

PORT(

DIFFICULTY : IN STD_LOGIC;

LVL: IN INTEGER := 0;

mode_sel: in std_logic;

CLR : in STD_LOGIC;

VIDON: in STD_LOGIC;

HC : in STD_LOGIC_VECTOR(10 downto 0);

VC : in STD_LOGIC_VECTOR(10 downto 0);

RED : out STD_LOGIC_VECTOR(3 downto 0);

GREEN : out STD_LOGIC_VECTOR(3 downto 0);

BLUE : out STD_LOGIC_VECTOR(3 downto 0)

);

end GAME_MODE;

---------------------------

architecture GAME_MODE of GAME_MODE is

signal GR_O,RED_O,BL_O : std_logic_vector(3 downto 0);

signal GR : std_logic_vector(103 downto 0);

signal BL :std_logic_vector(103 downto 0);

signal RE : std_logic_vector(103 downto 0);

signal MUX1,MUX2,MUX3,MUX4,MUX5,MUX6,MUX7,MUX8,MUX9,MUX10,MUX11,MUX12,MUX13,MUX14:
std_logic_vector(7 downto 0);

SIGNAL MUX1G, MUX2G, MUX3G, MUX4G,MUX5G : STD_LOGIC_VECTOR(7 DOWNTO 0);

SIGNAL MUX1D,MUX2D,MUX3D,MUX4D : STD_LOGIC_VECTOR(7 DOWNTO 0);

SIGNAL RED1,RED2,RED3,RED4,GRD1,GRD2,GRD3,GRD4,BLD1,BLD2,BLD3,BLD4 : STD_LOGIC_VECTOR(3
DOWNTO 0);

SIGNAL MUXE1,MUXE2,MUXE3,MUXE4,MUXE5 : STD_LOGIC_VECTOR(7 DOWNTO 0);

---------------------------

COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));
```

```
END COMPONENT;

begin

FU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0C",
R => 10,
C => 6,
HC => HC,
VC => VC,
RED => RE(71 downto 68),
GREEN => GR(71 downto 68),
BLUE => BL(71 downto 68)
);

FU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0E",
R => 10,
C => 14,
HC => HC,
VC => VC,
RED => RE(67 downto 64),
GREEN => GR(67 downto 64),
BLUE => BL(67 downto 64)
);

FU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"03",
R => 10,
C => 22,
HC => HC,
VC => VC,
RED => RE(63 downto 60),
GREEN => GR(63 downto 60),
BLUE => BL(63 downto 60)
);
FU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => 10,
C => 30,
HC => HC,
VC => VC,
RED => RE(59 downto 56),
GREEN => GR(59 downto 56),
BLUE => BL(59 downto 56)
);
FU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"23",
R => 10,
C => 38,
HC => HC,
VC => VC,
RED => RE(55 downto 52),
GREEN => GR(55 downto 52),
BLUE => BL(55 downto 52)
);
FU6 : CHAR_PLACER
PORT MAP(
MUX_IN => x"25",
R => 10,
C => 46,
HC => HC,
VC => VC,
RED => RE(51 downto 48),
GREEN => GR(51 downto 48),
BLUE => BL(51 downto 48)
);
FU7 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX1,
R => 10,
C => 54,
HC => HC,
VC => VC,
RED => RE(47 downto 44),
GREEN => GR(47 downto 44),
BLUE => BL(47 downto 44)
);
FU8 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX2,
R => 10,
C => 62,
HC => HC,
VC => VC,
RED => RE(43 downto 40),
GREEN => GR(43 downto 40),
BLUE => BL(43 downto 40)
);
FU9 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX3,
R => 10,
C => 70,
HC => HC,
VC => VC,
RED => RE(39 downto 36),
GREEN => GR(39 downto 36),
BLUE => BL(39 downto 36)
);
FU10 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX4,
R => 10,
C => 78,
HC => HC,
VC => VC,
RED => RE(35 downto 32),
GREEN => GR(35 downto 32),
BLUE => BL(35 downto 32)
);
FU11 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX5,
R => 10,
C => 86,
HC => HC,
VC => VC,
RED => RE(31 downto 28),
GREEN => GR(31 downto 28),
BLUE => BL(31 downto 28)
);
FU12 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX6,
R => 10,
C => 94,
HC => HC,
VC => VC,
RED => RE(27 downto 24),
GREEN => GR(27 downto 24),
BLUE => BL(27 downto 24)
);
FU13 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX9,
R => 10,
C => 102,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
```

```
);
FU14 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX10,
R => 10,
C => 110,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);
FU15 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX11,
R => 10,
C => 118,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);
FU16 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX12,
R => 10,
C => 126,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);
FU17 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX13,
R => 10,
C => 134,
HC => HC,
VC => VC,
RED => RE(7 downto 4),
GREEN => GR(7 downto 4),
BLUE => BL(7 downto 4)
);
FU18 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX14,
R => 10,
C => 142,
HC => HC,
VC => VC,
RED => RE(3 downto 0),
GREEN => GR(3 downto 0),
BLUE => BL(3 downto 0)
);
--------------------------------------
FUD19 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX1D,
R => 450,
C => 10,
HC => HC,
VC => VC,
RED => RED1,
GREEN => GRD1,
BLUE => BLD1
);
FUD20 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX2D,
R => 450,
C => 18,
HC => HC,
VC => VC,
RED => RED2,
GREEN => GRD2,
BLUE => BLD2
);
FUD21 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX3D,
R => 450,
C => 26,
HC => HC,
VC => VC,
RED => RED3,
GREEN => GRD3,
BLUE => BLD3
);
FUD22 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX4D,
R => 450,
C => 34,
HC => HC,
VC => VC,
RED => RED4,
GREEN => GRD4,
BLUE => BLD4
);
--------------------------------------
process(mode_sel,DIFFICULTY)
begin
case mode_sel is
when '1'=>    ---MEMORY GAME -- STAGE
MUX1 <= x"0C";
MUX2 <= x"04";
MUX3 <= x"0C";
MUX4 <= x"0E";
MUX5 <= x"11";
```

```
MUX6 <= x"18";
MUX9 <= x"25";
MUX10 <= x"06";
MUX11 <= x"00";
MUX12 <= x"0C";
MUX13 <= x"04";
MUX14 <= x"25";
MUX1G <= X"12";
MUX2G <= X"13";
MUX3G <= X"00";
MUX4G <= X"06";
MUX5G <= X"04";
MUX1D <= X"25";
MUX2D <= X"25";
MUX3D <= X"25";
MUX4D <= X"25";
MUXE1 <= X"12";
MUXE2 <= X"13";
MUXE3 <= X"00";
MUXE4 <= X"06";
MUXE5 <= X"04";
when '0' => -- WHACK A MOLE -- SCORE
MUX1 <= x"16";
MUX2 <= x"07";
MUX3 <= x"00";
MUX4 <= x"02";
MUX5 <= x"0A";
MUX6 <= x"25";
MUX9 <= x"00";
MUX10 <= x"25";
MUX11 <= x"0C";
MUX12 <= x"0E";
MUX13 <= x"0B";
MUX14 <= x"04";
MUX1G <= X"12";
MUX2G <= X"02";
MUX3G <= X"0E";
```

```
MUX4G <= X"11";
MUX5G <= X"04";
MUXE1 <= X"12";
MUXE2 <= X"02";
MUXE3 <= X"0E";
MUXE4 <= X"11";
MUXE5 <= X"04";
CASE DIFFICULTY IS
WHEN '1' =>
MUX1D <= X"04";
MUX2D <= X"00";
MUX3D <= X"12";
MUX4D <= X"18";
WHEN '0' =>
MUX1D <= X"07";
MUX2D <= X"00";
MUX3D <= X"11";
MUX4D <= X"03";
END CASE;
end case;
end process;
---------------------------------------------LEVEL INDICATOR--------------------------------------------
FU19 : CHAR_PLACER
PORT MAP(
MUX_IN => MUXE1,
R => 10,
C => 562,
HC => HC,
VC => VC,
RED => RE(75 downto 72),
GREEN => GR(75 downto 72),
BLUE => BL(75 downto 72)
);
FU20 : CHAR_PLACER
PORT MAP(
MUX_IN => MUXE2,
R => 10,
C => 570,

HC => HC,

VC => VC,

RED => RE(79 downto 76),

GREEN => GR(79 downto 76),

BLUE => BL(79 downto 76)

);

FU21 : CHAR_PLACER

PORT MAP(

MUX_IN => MUXE3,

R => 10,

C => 578,

HC => HC,

VC => VC,

RED => RE(83 downto 80),

GREEN => GR(83 downto 80),

BLUE => BL(83 downto 80)

);

FU22 : CHAR_PLACER

PORT MAP(

MUX_IN => MUXE4,

R => 10,

C => 586,

HC => HC,

VC => VC,

RED => RE(87 downto 84),

GREEN => GR(87 downto 84),

BLUE => BL(87 downto 84)

);

FU23 : CHAR_PLACER

PORT MAP(

MUX_IN => MUXE5,

R => 10,

C => 594,

HC => HC,

VC => VC,

RED => RE(91 downto 88),
GREEN => GR(91 downto 88),
BLUE => BL(91 downto 88)
);
FU24 : CHAR_PLACER
PORT MAP(
MUX_IN => x"23",
R => 10,
C => 602,
HC => HC,
VC => VC,
RED => RE(95 downto 92),
GREEN => GR(95 downto 92),
BLUE => BL(95 downto 92)
);
FU25 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX7,
R => 10,
C => 610,
HC => HC,
VC => VC,
RED => RE(99 downto 96),
GREEN => GR(99 downto 96),
BLUE => BL(99 downto 96)
);
FU26 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX8,
R => 10,
C => 618,
HC => HC,
VC => VC,
RED => RE(103 downto 100),
GREEN => GR(103 downto 100),
BLUE => BL(103 downto 100)
);
-----------------------------------------------------------------------------------------------
```

```
PROCESS(LVL)

VARIABLE M : integer := (LVL mod 10);

VARIABLE K : integer := ((LVL - (LVL mod 10))/10);

BEGIN

IF K = 1 THEN

MUX7 <= x"1A";

ELSIF K = 2 THEN

MUX7 <= x"1B";

ELSIF K = 3 THEN

MUX7 <= x"1C";

ELSIF K = 4 THEN

MUX7 <= x"1D";

ELSIF K = 5 THEN

MUX7 <= x"1E";

ELSIF K = 6 THEN

MUX7 <= x"1F";

ELSIF K = 7 THEN

MUX7 <= x"20";

ELSIF K = 8 THEN

MUX7 <= x"21";

ELSIF K = 9 THEN

MUX7 <= x"22";

ELSIF K = 0 THEN

MUX7 <= x"24";

END IF;


IF M = 1 THEN

MUX8 <= x"1A";

ELSIF M = 2 THEN

MUX8 <= x"1B";

ELSIF M = 3 THEN

MUX8 <= x"1C";

ELSIF M = 4 THEN

MUX8 <= x"1D";

ELSIF M = 5 THEN

MUX8 <= x"1E";

ELSIF M = 6 THEN
```

```
MUX8 <= x"1F";

ELSIF M = 7 THEN

MUX8 <= x"20";

ELSIF M = 8 THEN

MUX8 <= x"21";

ELSIF M = 9 THEN

MUX8 <= x"22";

ELSIF M = 0 THEN

MUX8 <= x"24";

END IF;

END PROCESS;

----------------------------------------------------------------------------------------------------

RED_O <=

RE(103 DOWNTO 100) OR RE(99 DOWNTO 96) OR RE(95 DOWNTO 92) OR

RE(91 downto 88) OR RE(87 DOWNTO 84)OR RE(83 DOWNTO 80) OR RE(79 DOWNTO 76) OR RE(75 DOWNTO 72) OR

RE(71 downto 68) OR RE(67 DOWNTO 64)OR RE(63 DOWNTO 60) OR RE(59 DOWNTO 56) OR RE(55 DOWNTO 52) OR

RE(51 downto 48) OR RE(47 DOWNTO 44)OR RE(43 DOWNTO 40) OR RE(39 DOWNTO 36) OR RE(35 DOWNTO 32) OR

RE(31 downto 28) OR RE(27 DOWNTO 24)OR RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR

RE(11 downto 8) OR RE(7 DOWNTO 4)OR RE(3 DOWNTO 0) OR RED1 OR RED2 OR RED3 OR RED4;

GR_O <=

GR(103 DOWNTO 100) OR GR(99 DOWNTO 96) OR GR(95 DOWNTO 92) OR

GR(91 downto 88) OR GR(87 DOWNTO 84)OR GR(83 DOWNTO 80) OR GR(79 DOWNTO 76) OR GR(75 DOWNTO 72)
OR

GR(71 downto 68) OR GR(67 DOWNTO 64)OR GR(63 DOWNTO 60) OR GR(59 DOWNTO 56) OR GR(55 DOWNTO 52)
OR

GR(51 downto 48) OR GR(47 DOWNTO 44)OR GR(43 DOWNTO 40) OR GR(39 DOWNTO 36) OR GR(35 DOWNTO 32)
OR

GR(31 downto 28) OR GR(27 DOWNTO 24)OR GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12)
OR

GR(11 downto 8) OR GR(7 DOWNTO 4)OR GR(3 DOWNTO 0) OR GRD1 OR GRD2 OR GRD3 OR GRD4;

BL_O <=

BL(103 DOWNTO 100) OR BL(99 DOWNTO 96) OR BL(95 DOWNTO 92) OR

BL(91 downto 88) OR BL(87 DOWNTO 84)OR BL(83 DOWNTO 80) OR BL(79 DOWNTO 76) OR BL(75 DOWNTO 72) OR

BL(71 downto 68) OR BL(67 DOWNTO 64)OR BL(63 DOWNTO 60) OR BL(59 DOWNTO 56) OR BL(55 DOWNTO 52) OR

BL(51 downto 48) OR BL(47 DOWNTO 44)OR BL(43 DOWNTO 40) OR BL(39 DOWNTO 36) OR BL(35 DOWNTO 32) OR

BL(31 downto 28) OR BL(27 DOWNTO 24)OR BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR

BL(11 downto 8) OR BL(7 DOWNTO 4)OR BL(3 DOWNTO 0) OR BLD1 OR BLD2 OR BLD3 OR BLD4 ;


process(VIDON,CLR,RED_O,GR_O,BL_O)
```

```
begin
if VIDON = '1' then
if CLR = '1' then
RED <= "0000";
GREEN <= "0000";
BLUE <= "0000";
else
RED <= RED_O;
GREEN <= GR_O;
BLUE <= BL_O;
end if;
end if;
end process;
end GAME_MODE;
```

CHAR_PLACER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity CHAR_PLACER is
   Port ( MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
          R : in integer;
          C : in integer;
          HC : in STD_LOGIC_VECTOR (10 downto 0);
          VC : in STD_LOGIC_VECTOR (10 downto 0);
          RED : out STD_LOGIC_VECTOR (3 downto 0);
          GREEN : out STD_LOGIC_VECTOR (3 downto 0);
          BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end CHAR_PLACER;


architecture CHAR_PLACER of CHAR_PLACER is
signal ADDR : STD_LOGIC_VECTOR(3 downto 0);
signal ADDRX : STD_LOGIC_VECTOR(3 downto 0);
signal M : STD_LOGIC_VECTOR(0 to 7);
signal spriteon , Re,Gr,Bl : std_logic;
signal rom_addr, rom_pix : std_logic_vector(10 downto 0);
constant W :integer := 8;
constant H: integer := 16;

COMPONENT ALNUM
PORT(
S: in STD_LOGIC_VECTOR(7 downto 0);
ADDR: in STD_LOGIC_VECTOR(3 downto 0);
M: out STD_LOGIC_VECTOR(0 to 7)
);
END COMPONENT;

begin

ALPHA : ALNUM
PORT MAP(
S => MUX_IN,
ADDR => ADDR,
M => M
);
rom_addr <= VC - R;
rom_pix <= HC - C;
ADDR <= rom_addr(3 downto 0);

SPRITEON <= '1' when (((HC >= C) AND (HC < C + W)) AND ((VC >= R) AND (VC < R + H))) else '0';

process(Re,Gr,Bl,SPRITEON,rom_pix,M)
variable j : integer;
begin
   RED <= "0000";
   GREEN <= "0000";
   BLUE <= "0000";
IF SPRITEON = '1' THEN
j := conv_integer(rom_pix);

Re <= M(j);
Gr <= M(j);
Bl <= M(j);
```

```
RED <= Re & Re & Re & Re;

GREEN <= Gr & Gr & Gr & Gr;

BLUE <= Bl & Bl & Bl & Bl;


end if;

end process;

end CHAR_PLACER;
```

ALNUM

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ALNUM is

PORT(

S: in STD_LOGIC_VECTOR(7 downto 0);

ADDR: in STD_LOGIC_VECTOR(3 downto 0);

M: out STD_LOGIC_VECTOR(0 to 7)

);


end ALNUM;


architecture ALNUM of ALNUM is

type rom_array is array (0 to 15) of STD_LOGIC_VECTOR(7 downto 0);


signal use_rom : rom_array;


constant roma : rom_array := (

   "00000000", -- 0

   "00000000", -- 1

   "00111100", -- 2

   "01100110", -- 3

   "01100110", -- 4

   "01000010", -- 5  ****

   "01000010", -- 6     **

   "01111110", -- 7  *****

   "01100110", -- 8 **  **

   "01100110", -- 9 **  **
```

```
    "01100110", -- a **  **
    "01100110", -- b  *** **
    "00000000", -- c
    "00000000", -- d
    "00000000", -- e
    "00000000" -- f
);

constant romb : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111100", -- 2 ******
   "01100110", -- 3  **  **
   "01100110", -- 4  **  **
   "01100110", -- 5  **  **
   "01111100", -- 6  *****
   "01100110", -- 7  **  **
   "01100110", -- 8  **  **
   "01100110", -- 9  **  **
   "01100110", -- a  **  **
   "11111100", -- b ******
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romc : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00111100", -- 2   ****
   "01100110", -- 3  **  **
   "11000010", -- 4 **    *
   "11000000", -- 5 **
   "11000000", -- 6 **
   "11000000", -- 7 **
   "11000000", -- 8 **
   "11000010", -- 9 **     *
   "01100110", -- a  **  **
   "00111100", -- b   ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);

constant romd : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111000", -- 2 *****
   "01101100", -- 3  ** **
   "01100110", -- 4  **  **
   "01100110", -- 5  **  **
   "01100110", -- 6  **  **
   "01100110", -- 7  **  **
   "01100110", -- 8  **  **
   "01100110", -- 9  **  **
   "01101100", -- a  ** **
   "11111000", -- b *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);

constant rome : rom_array := (

   "00000000", -- 0
   "00000000", -- 1
   "11111110", -- 2 *******
   "01100110", -- 3  **  **
   "01100010", -- 4  **   *
   "01101000", -- 5  ** *
   "01111000", -- 6  ****
   "01101000", -- 7  ** *
   "01100000", -- 8  **
   "01100010", -- 9  **   *
   "01100110", -- a  **  **
   "11111110", -- b *******
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);



constant romf : rom_array := (

   "00000000", -- 0
   "00000000", -- 1
   "11111110", -- 2 *******
   "01100110", -- 3  **  **
   "01100010", -- 4  **   *
   "01101000", -- 5  ** *
   "01111000", -- 6  ****
   "01101000", -- 7  ** *
   "01100000", -- 8  **
   "01100000", -- 9  **
   "01100000", -- a  **
   "11110000", -- b ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);
```

```
constant romg : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00111100", -- 2   ****
   "01100110", -- 3  **  **
   "11000010", -- 4 **    *
   "11000000", -- 5 **
   "11000000", -- 6 **
   "11011110", -- 7 ** ****
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "01100110", -- a  **  **
   "00111010", -- b   *** *
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);



constant romh : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000110", -- 2 **   **
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "11111110", -- 6 *******
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "11000110", -- b **   **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);



constant romi : rom_array := (


   "00000000", -- 0
   "00000000", -- 1
   "00111100", -- 2   ****
   "00011000", -- 3    **
   "00011000", -- 4    **
   "00011000", -- 5    **
   "00011000", -- 6    **
   "00011000", -- 7    **
   "00011000", -- 8    **
   "00011000", -- 9    **
   "00011000", -- a    **
   "00111100", -- b   ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);



constant romj : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00011110", -- 2    ****
   "00001100", -- 3     **
   "00001100", -- 4     **
   "00001100", -- 5     **
   "00001100", -- 6     **
   "00001100", -- 7     **
   "11001100", -- 8 **  **
   "11001100", -- 9 **  **
   "11001100", -- a **  **
   "01111000", -- b  ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f

);

constant romk : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11100110", -- 2 ***  **
   "01100110", -- 3  **  **
   "01100110", -- 4  **  **
   "01101100", -- 5  ** **
   "01111000", -- 6  ****
   "01111000", -- 7  ****
   "01101100", -- 8  ** **
   "01100110", -- 9  **  **
   "01100110", -- a  **  **
   "11100110", -- b ***  **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant roml : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11110000", -- 2 ****
   "01100000", -- 3  **
   "01100000", -- 4  **
   "01100000", -- 5  **
   "01100000", -- 6  **
   "01100000", -- 7  **
   "01100000", -- 8  **
   "01100010", -- 9  **   *
   "01100110", -- a  **  **
   "11111110", -- b *******
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romm : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000011", -- 2 **    **
   "11100111", -- 3 ***  ***
   "11111111", -- 4 ********
   "11111111", -- 5 ********
   "11011011", -- 6 ** ** **
   "11000011", -- 7 **    **
   "11000011", -- 8 **    **
   "11000011", -- 9 **    **
   "11000011", -- a **    **
   "11000011", -- b **    **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romn : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000110", -- 2 **   **
   "11100110", -- 3 ***  **
   "11110110", -- 4 **** **
   "11111110", -- 5 *******
   "11011110", -- 6 ** ****
   "11001110", -- 7 **  ***
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "11000110", -- b **   **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romo : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "11000110", -- 6 **   **
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romp : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111100", -- 2 ******
   "01100110", -- 3  **  **
   "01100110", -- 4  **  **
   "01100110", -- 5  **  **
   "01111100", -- 6  *****
   "01100000", -- 7  **
   "01100000", -- 8  **
   "01100000", -- 9  **
   "01100000", -- a  **
   "11110000", -- b ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romq : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "11000110", -- 6 **   **
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11010110", -- 9 ** * **
   "11011110", -- a ** ****
   "01111100", -- b  *****
   "00001100", -- c     **
   "00001110", -- d     ***
   "00000000", -- e
   "00000000" -- f
);

constant romr : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111100", -- 2 ******
   "01100110", -- 3  **  **
   "01100110", -- 4  **  **
   "01100110", -- 5  **  **
   "01111100", -- 6  *****
   "01101100", -- 7  ** **
   "01100110", -- 8  **  **
   "01100110", -- 9  **  **
   "01100110", -- a  **  **
   "11100110", -- b ***  **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant roms : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "01100000", -- 5  **
   "00111000", -- 6   ***
   "00001100", -- 7     **
   "00000110", -- 8      **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romt : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111111", -- 2 ********
   "11011011", -- 3 ** ** **
   "10011001", -- 4 *  **  *
   "00011000", -- 5    **
   "00011000", -- 6    **
   "00011000", -- 7    **
   "00011000", -- 8    **
   "00011000", -- 9    **
   "00011000", -- a    **
   "00111100", -- b   ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romu : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000110", -- 2 **   **
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "11000110", -- 6 **   **
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romv : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000011", -- 2 **    **
   "11000011", -- 3 **    **
   "11000011", -- 4 **    **
   "11000011", -- 5 **    **
   "11000011", -- 6 **    **
   "11000011", -- 7 **    **
   "11000011", -- 8 **    **
   "01100110", -- 9  **  **
   "00111100", -- a   ****
   "00011000", -- b    **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romw : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000011", -- 2 **    **
   "11000011", -- 3 **    **
   "11000011", -- 4 **    **
   "11000011", -- 5 **    **
   "11000011", -- 6 **    **
   "11011011", -- 7 ** ** **
   "11011011", -- 8 ** ** **
   "11111111", -- 9 ********
   "01100110", -- a  **  **
   "01100110", -- b  **  **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romx : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000011", -- 2 **    **
   "11000011", -- 3 **    **
   "01100110", -- 4  **  **
   "00111100", -- 5   ****
   "00011000", -- 6    **
   "00011000", -- 7    **
   "00111100", -- 8   ****
   "01100110", -- 9  **  **
   "11000011", -- a **    **
   "11000011", -- b **    **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romy : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11000011", -- 2 **    **
   "11000011", -- 3 **    **
   "11000011", -- 4 **    **
   "01100110", -- 5  **  **
   "00111100", -- 6   ****
   "00011000", -- 7    **
   "00011000", -- 8    **
   "00011000", -- 9    **
   "00011000", -- a    **
   "00111100", -- b   ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romz : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111111", -- 2 ********
   "11000011", -- 3 **    **
   "10000110", -- 4 *    **
   "00001100", -- 5     **
   "00011000", -- 6    **
   "00110000", -- 7   **
   "01100000", -- 8  **
   "11000001", -- 9 **     *
   "11000011", -- a **    **
   "11111111", -- b ********
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom0 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11001110", -- 5 **  ***
   "11011110", -- 6 ** ****
   "11110110", -- 7 **** **
   "11100110", -- 8 ***  **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);
```

```
constant rom1 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00011000", -- 2
   "00111000", -- 3
   "01111000", -- 4    **
   "00011000", -- 5   ***
   "00011000", -- 6  ****
   "00011000", -- 7    **
   "00011000", -- 8    **
   "00011000", -- 9    **
   "00011000", -- a    **
   "01111110", -- b    **
   "00000000", -- c    **
   "00000000", -- d  ******
   "00000000", -- e
   "00000000" -- f
);

constant rom2 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "00000110", -- 4      **
   "00001100", -- 5     **
   "00011000", -- 6    **
   "00110000", -- 7   **
   "01100000", -- 8  **
   "11000000", -- 9 **
   "11000110", -- a **   **
   "11111110", -- b *******
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);
```

```
constant rom3 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "00000110", -- 4      **
   "00000110", -- 5      **
   "00111100", -- 6   ****
   "00000110", -- 7      **
   "00000110", -- 8      **
   "00000110", -- 9      **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom4 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00001100", -- 2     **
   "00011100", -- 3    ***
   "00111100", -- 4   ****
   "01101100", -- 5  ** **
   "11001100", -- 6 **  **
   "11111110", -- 7 *******
   "00001100", -- 8     **
   "00001100", -- 9     **
   "00001100", -- a     **
   "00011110", -- b    ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
```

```
);

constant rom5 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111110", -- 2 *******
   "11000000", -- 3 **
   "11000000", -- 4 **
   "11000000", -- 5 **
   "11111100", -- 6 ******
   "00000110", -- 7      **
   "00000110", -- 8      **
   "00000110", -- 9      **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom6 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00111000", -- 2   ***
   "01100000", -- 3  **
   "11000000", -- 4 **
   "11000000", -- 5 **
   "11111100", -- 6 ******
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom7 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "11111110", -- 2 *******
   "11000110", -- 3 **   **
   "00000110", -- 4      **
   "00000110", -- 5      **
   "00001100", -- 6     **
   "00011000", -- 7    **
   "00110000", -- 8   **
   "00110000", -- 9   **
   "00110000", -- a   **
   "00110000", -- b   **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom8 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "01111100", -- 6  *****
   "11000110", -- 7 **   **
   "11000110", -- 8 **   **
   "11000110", -- 9 **   **
   "11000110", -- a **   **
   "01111100", -- b  *****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant rom9 : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "01111100", -- 2  *****
   "11000110", -- 3 **   **
   "11000110", -- 4 **   **
   "11000110", -- 5 **   **
   "01111110", -- 6  ******
   "00000110", -- 7      **
   "00000110", -- 8      **
   "00000110", -- 9      **
   "00001100", -- a     **
   "01111000", -- b  ****
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant romdoubledot : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00000000", -- 2
   "00000000", -- 3
   "00011000", -- 4    **
   "00011000", -- 5    **
   "00000000", -- 6
   "00000000", -- 7
   "00000000", -- 8
   "00011000", -- 9    **
   "00011000", -- a    **
   "00000000", -- b
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant nullrom : rom_array := (
    "00000000",--0
    "00000000",-- 1
    "00000000",-- 2
    "00000000",-- 3
    "00000000",-- 4
    "00000000",-- 5
    "00000000",-- 6
    "00000000",-- 7
    "00000000",-- 8
    "00000000",-- 9
    "00000000",-- a
    "00000000",-- b
    "00000000",-- c
    "00000000",-- d
    "00000000",-- e
    "00000000"-- f
);

constant noterom : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00111111", -- 2   ******
   "00110011", -- 3   **  **
   "00111111", -- 4   ******
   "00110000", -- 5   **
   "00110000", -- 6   **
   "00110000", -- 7   **
   "00110000", -- 8   **
   "01110000", -- 9  ***
   "11110000", -- a ****
   "11100000", -- b ***
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant uprom : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00011000", -- 2    **
   "00111100", -- 3   ****
   "01111110", -- 4  ******
   "00011000", -- 5    **
   "00011000", -- 6    **
   "00011000", -- 7    **
   "00011000", -- 8    **
   "00011000", -- 9    **
   "00011000", -- a    **
   "00011000", -- b    **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
);

constant downrom : rom_array := (
   "00000000", -- 0
   "00000000", -- 1
   "00011000", -- 2    **
   "00011000", -- 3    **
   "00011000", -- 4    **
   "00011000", -- 5    **
   "00011000", -- 6    **
   "00011000", -- 7    **
   "00011000", -- 8    **
   "01111110", -- 9  ******
   "00111100", -- a   ****
   "00011000", -- b    **
   "00000000", -- c
   "00000000", -- d
   "00000000", -- e
   "00000000" -- f
   );

begin

multiplexing_alphanumeric_values: process(S)
begin
    case S is
        when x"00" => use_rom <= roma; --A
        when x"01" => use_rom <= romb; --B
        when x"02" => use_rom <= romc; --C
        when x"03" => use_rom <= romd; --D
        when x"04" => use_rom <= rome; --E
        when x"05" => use_rom <= romf; --F
        when x"06" => use_rom <= romg; --G
        when x"07" => use_rom <= romh; --H
        when x"08" => use_rom <= romi; --I
        when x"09" => use_rom <= romj; --J
        when x"0A" => use_rom <= romk; --K
        when x"0B" => use_rom <= roml; --L
        when x"0C" => use_rom <= romm; --M
        when x"0D" => use_rom <= romn; --N
        when x"0E" => use_rom <= romo; --O
        when x"0F" => use_rom <= romp; --P
        when x"10" => use_rom <= romq; --Q
        when x"11" => use_rom <= romr; --R
        when x"12" => use_rom <= roms; --S
        when x"13" => use_rom <= romt; --T
        when x"14" => use_rom <= romu; --U
        when x"15" => use_rom <= romv; --V
        when x"16" => use_rom <= romw; --W
        when x"17" => use_rom <= romx; --X
        when x"18" => use_rom <= romy; --Y
        when x"19" => use_rom <= romz; --Z
        when x"1A" => use_rom <= rom1; --1
        when x"1B" => use_rom <= rom2; --2
        when x"1C" => use_rom <= rom3; --3
        when x"1D" => use_rom <= rom4; --4
        when x"1E" => use_rom <= rom5; --5
        when x"1F" => use_rom <= rom6; --6
        when x"20" => use_rom <= rom7; --7
        when x"21" => use_rom <= rom8; --8
        when x"22" => use_rom <= rom9; --9
        when x"23" => use_rom <= romdoubledot;-- :
        when x"24" => use_rom <= rom0; --0
        when x"25" => use_rom <= nullrom;
        when x"26" => use_rom <= noterom;
        when x"27" => use_rom <= uprom;
        when x"28" => use_rom <= downrom;
        when others => use_rom <= nullrom;

end case;
end process;



process(use_rom,ADDR)
variable j: integer;
begin
j := conv_integer(ADDR);
M <= use_rom(j);
end process;
end ALNUM;
```

GAME_SOUNDS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity GAME_SOUNDS is
PORT(
audio_sel: in std_logic;
CLR : in STD_LOGIC;
```

```
VIDON: in STD_LOGIC;
HC : in STD_LOGIC_VECTOR(10 downto 0);
VC : in STD_LOGIC_VECTOR(10 downto 0);
RED : out STD_LOGIC_VECTOR(3 downto 0);
GREEN : out STD_LOGIC_VECTOR(3 downto 0);
BLUE : out STD_LOGIC_VECTOR(3 downto 0)
);
end GAME_SOUNDS;

architecture GAME_SOUNDS of GAME_SOUNDS is
signal GR_O,RED_O,BL_O : std_logic_vector(3 downto 0);
signal GR : std_logic_vector(71 downto 0);
signal BL :std_logic_vector(71 downto 0);
signal RE : std_logic_vector(71 downto 0);
signal MUX1,MUX2,MUX3: std_logic_vector(7 downto 0);
COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin
FU0 : CHAR_PLACER
PORT MAP(
MUX_IN => x"26",
R => 462,
C => 538,
HC => HC,
VC => VC,
RED => RE(31 downto 28),
GREEN => GR(31 downto 28),
BLUE => BL(31 downto 28)
);

FU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => 462,
C => 546,
HC => HC,
VC => VC,
RED => RE(71 downto 68),
GREEN => GR(71 downto 68),
BLUE => BL(71 downto 68)
);

FU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"14",
R => 462,
C => 554,
HC => HC,
VC => VC,
RED => RE(67 downto 64),
GREEN => GR(67 downto 64),
BLUE => BL(67 downto 64)
);

FU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"03",
R => 462,
C => 562,
HC => HC,
VC => VC,
RED => RE(63 downto 60),
GREEN => GR(63 downto 60),
BLUE => BL(63 downto 60)
```

```
);
FU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"08",
R => 462,
C => 570,
HC => HC,
VC => VC,
RED => RE(59 downto 56),
GREEN => GR(59 downto 56),
BLUE => BL(59 downto 56)
);
FU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0E",
R => 462,
C => 578,
HC => HC,
VC => VC,
RED => RE(55 downto 52),
GREEN => GR(55 downto 52),
BLUE => BL(55 downto 52)
);
FU6 : CHAR_PLACER
PORT MAP(
MUX_IN => x"23",
R => 462,
C => 586,
HC => HC,
VC => VC,
RED => RE(51 downto 48),
GREEN => GR(51 downto 48),
BLUE => BL(51 downto 48)
);
FU7 : CHAR_PLACER
PORT MAP(
MUX_IN => x"25",
R => 462,
C => 594,
HC => HC,
VC => VC,
RED => RE(47 downto 44),
GREEN => GR(47 downto 44),
BLUE => BL(47 downto 44)
);
FU8 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX1,
R => 462,
C => 602,
HC => HC,
VC => VC,
RED => RE(43 downto 40),
GREEN => GR(43 downto 40),
BLUE => BL(43 downto 40)
);
FU9 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX2,
R => 462,
C => 610,
HC => HC,
VC => VC,
RED => RE(39 downto 36),
GREEN => GR(39 downto 36),
BLUE => BL(39 downto 36)
);
FU10 : CHAR_PLACER
PORT MAP(
MUX_IN => MUX3,
R => 462,
C => 618,
HC => HC,
VC => VC,
RED => RE(35 downto 32),

GREEN => GR(35 downto 32),

BLUE => BL(35 downto 32)

);


process(audio_sel)

begin

case audio_sel is

when '1'=>   ------------ AUDIO ON

MUX1 <= x"0E";

MUX2 <= x"0D";

MUX3 <= x"25";


when '0' => ---------- AUDIO OFF

MUX1 <= x"0E";

MUX2 <= x"05";

MUX3 <= x"05";




end case;

end process;

RED_O <=

RE(71 downto 68) OR RE(67 DOWNTO 64)OR RE(63 DOWNTO 60) OR RE(59 DOWNTO 56) OR RE(55 DOWNTO 52) OR

RE(51 downto 48) OR RE(47 DOWNTO 44)OR RE(43 DOWNTO 40) OR RE(39 DOWNTO 36) OR RE(35 DOWNTO 32) OR
RE(31 DOWNTO 28);

GR_O <=

GR(71 downto 68) OR GR(67 DOWNTO 64)OR GR(63 DOWNTO 60) OR GR(59 DOWNTO 56) OR GR(55 DOWNTO 52)
OR

GR(51 downto 48) OR GR(47 DOWNTO 44)OR GR(43 DOWNTO 40) OR GR(39 DOWNTO 36) OR GR(35 DOWNTO 32)OR
GR(31 DOWNTO 28);

BL_O <=

BL(71 downto 68) OR BL(67 DOWNTO 64)OR BL(63 DOWNTO 60) OR BL(59 DOWNTO 56) OR BL(55 DOWNTO 52) OR

BL(51 downto 48) OR BL(47 DOWNTO 44)OR BL(43 DOWNTO 40) OR BL(39 DOWNTO 36) OR BL(35 DOWNTO 32) OR
BL(31 DOWNTO 28);


process(VIDON,CLR,RED_O,BL_O,GR_O)

begin

if VIDON = '1' then

if CLR = '1' then
```

```
RED <= "0000";
GREEN <= "0000";
BLUE <= "0000";
else
RED <= RED_O;
GREEN <= GR_O;
BLUE <= BL_O;
end if;
end if;
end process;
end GAME_SOUNDS;
```

SCORE_COUNT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
--------------------------------
entity SCORE_COUNT is
PORT(
DIF : IN STD_LOGIC;
CLK : IN STD_LOGIC;
RESET : IN STD_LOGIC;
BTN : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
GAME_OVER : OUT STD_LOGIC;
SQUARES : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
SCORE : OUT INTEGER:= 0
);
end SCORE_COUNT;
--------------------------------
architecture Behavioral of SCORE_COUNT is
TYPE STATE_TYPE IS (S_CATCH,S_MISS,S_OVER,S_INIT,S_WAIT);
SIGNAL MOLE : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL STATE : STATE_TYPE;
SIGNAL SCOREE : INTEGER := 5;
SIGNAL RAND : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL BTN0P,BTN0R,BTN1P,BTN1R,BTN2P,BTN2R,BTN3P,BTN3R,BTN4P,BTN4R : STD_LOGIC;
SIGNAL CLK_G,CLK_W : STD_LOGIC;
SIGNAL PUSHED : STD_LOGIC;
-----------------------------------
COMPONENT LFSR4 IS
PORT(
DIF : IN STD_LOGIC;
CLK : IN STD_LOGIC;
RAND_COLOR : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
RAND_NUMB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END COMPONENT;
----------------------
COMPONENT PUSH_RELEASE IS
PORT(
CLK : IN STD_LOGIC;
BTN_IN : IN STD_LOGIC;
BTN_OUT_P : OUT STD_LOGIC;
BTN_OUT_R : OUT STD_LOGIC
);
END COMPONENT;

---------------------------
BEGIN
RANDOM_GEN : LFSR4
PORT MAP(
DIF => DIF,
CLK => CLK,
RAND_COLOR => MOLE,
RAND_NUMB => RAND
);
-----------------------------
BTN0 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(0),BTN_OUT_P => BTN0P,BTN_OUT_R => BTN0R
);
BTN1 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(1),BTN_OUT_P => BTN1P,BTN_OUT_R => BTN1R
);
BTN2 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(2),BTN_OUT_P => BTN2P,BTN_OUT_R => BTN2R
);
BTN3 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(3),BTN_OUT_P => BTN3P,BTN_OUT_R => BTN3R
);
```

```
BTN4 : PUSH_RELEASE PORT MAP(CLK => CLK, BTN_IN => BTN(4),BTN_OUT_P => BTN4P,BTN_OUT_R => BTN4R
);

PUSHED <= BTN0P OR BTN1P OR BTN2P OR BTN3P OR BTN4P;

SQUARES <= MOLE;

SCORE <= SCOREE;

----------------------

PROCESS(CLK)

VARIABLE CAUGHT : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN

IF RESET = '1' THEN

STATE <= S_INIT;

ELSIF RISING_EDGE(CLK) THEN

CASE STATE IS


WHEN S_INIT =>

GAME_OVER <= '0';

SCOREE <= 10;

IF BTN0P = '1' THEN

STATE <= S_WAIT;

ELSE

STATE <= S_INIT;

END IF;

-----------------------------------

WHEN S_WAIT =>

IF SCOREE > 0 THEN

IF PUSHED /= '0' THEN

IF BTN = MOLE AND CAUGHT /= RAND THEN

STATE <= S_CATCH;

ELSIF BTN /= MOLE THEN

STATE <= S_MISS;

END IF;

ELSE

STATE <= S_WAIT;

END IF;

ELSE

STATE <= S_OVER;

END IF;
```

```
------------------------------------
WHEN S_MISS =>
SCOREE <= SCOREE - 1;
CAUGHT := RAND;
STATE <= S_WAIT;
------------------------------------
WHEN S_CATCH =>
SCOREE <= SCOREE + 1;
CAUGHT := RAND;
STATE <= S_WAIT;
------------------------------------
WHEN S_OVER =>
GAME_OVER <= '1';
STATE <= S_OVER;
------------------------------------
END CASE;
END IF;
END PROCESS;
end Behavioral;
```

LFSR4

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
---------------------------
ENTITY LFSR4 is
PORT(
DIF : IN STD_LOGIC;
CLK : IN STD_LOGIC;
RAND_COLOR : OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
RAND_NUMB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END LFSR4;
---------------------------
ARCHITECTURE BEHAVIORAL OF LFSR4 IS
SIGNAL RAND_CLK : STD_LOGIC;
SIGNAL RANDOM_NUM : STD_LOGIC_VECTOR(3 DOWNTO 0);
TYPE ARRAY_TYPE IS ARRAY (0 to 4) of STD_LOGIC_VECTOR(4 DOWNTO 0);
```

```
CONSTANT COLOR_PICK : ARRAY_TYPE :=(
"10000",
"01000",
"00100",
"00010",
"00001"
);
---------------------------
COMPONENT RANDOM_CLK_DIV IS
PORT(
DIF : IN STD_LOGIC;
CLK : IN STD_LOGIC;
CLK_O : OUT STD_LOGIC
);
END COMPONENT;
---------------------------
BEGIN
---------------------------
RANDOM_CLOCK : RANDOM_CLK_DIV
PORT MAP(
DIF => DIF,
CLK => CLK,
CLK_O => RAND_CLK
);
---------------------------
PROCESS(RAND_CLK)
VARIABLE RAND_TEMP : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";
VARIABLE TEMP : STD_LOGIC := '0';
BEGIN
IF RISING_EDGE(RAND_CLK) THEN
TEMP := RAND_TEMP(3) XOR RAND_TEMP(2);
RAND_TEMP(3 DOWNTO 1) := RAND_TEMP(2 DOWNTO 0);
RAND_TEMP(0) := TEMP;
END IF;
RANDOM_NUM <= RAND_TEMP;
END PROCESS;
---------------------------
```

```
PROCESS(RANDOM_NUM,CLK)

BEGIN

IF RISING_EDGE(CLK) THEN

CASE RANDOM_NUM IS

WHEN X"1" => RAND_COLOR <= COLOR_PICK(0);

WHEN X"2" => RAND_COLOR <= COLOR_PICK(1);

WHEN X"3" => RAND_COLOR <= COLOR_PICK(2);

WHEN X"4" => RAND_COLOR <= COLOR_PICK(3);

WHEN X"5" => RAND_COLOR <= COLOR_PICK(4);

WHEN X"6" => RAND_COLOR <= COLOR_PICK(0);

WHEN X"7" => RAND_COLOR <= COLOR_PICK(1);

WHEN X"8" => RAND_COLOR <= COLOR_PICK(2);

WHEN X"9" => RAND_COLOR <= COLOR_PICK(3);

WHEN X"A" => RAND_COLOR <= COLOR_PICK(4);

WHEN X"B" => RAND_COLOR <= COLOR_PICK(0);

WHEN X"C" => RAND_COLOR <= COLOR_PICK(1);

WHEN X"D" => RAND_COLOR <= COLOR_PICK(2);

WHEN X"E" => RAND_COLOR <= COLOR_PICK(3);

WHEN X"F" => RAND_COLOR <= COLOR_PICK(4);

WHEN OTHERS => NULL;

END CASE;

END IF;

END PROCESS;

---------------------------

RAND_NUMB <= RANDOM_NUM;

END BEHAVIORAL;
```

RANDOM_CLK_DIV

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

------------------------------

ENTITY RANDOM_CLK_DIV is

PORT (

DIF : IN STD_LOGIC;

CLK : in std_logic;

CLK_O : out std_logic);
```

```
end RANDOM_CLK_DIV;
-----------------------------
ARCHITECTURE RANDOM_CLK_DIV of RANDOM_CLK_DIV is
CONSTANT COUNT1 : INTEGER := (17500000);
CONSTANT COUNT2 : INTEGER := (25000000);
SIGNAL COUNT : INTEGER := (17500000);
SIGNAL TMP : std_logic := '0';
------------------------------
BEGIN
PROCESS(CLK,DIF)
BEGIN
IF RISING_EDGE(CLK) THEN
CASE DIF IS
WHEN '0' => COUNT <= COUNT1;
WHEN '1' => COUNT <= COUNT2;
END CASE;
END IF;
END PROCESS;
DIV_CLK: PROCESS(CLK,TMP)
VARIABLE DIV_CNT : integer := 0;
BEGIN
IF (rising_edge(clk)) THEN
IF (DIV_CNT = COUNT) THEN
TMP <= NOT TMP;
DIV_CNT := 0;
ELSIF DIV_CNT < COUNT THEN
DIV_CNT := DIV_CNT + 1;
ELSE
DIV_CNT := 0;
END IF;
END IF;
CLK_O <= TMP;
END PROCESS DIV_CLK;
END RANDOM_CLK_DIV;
```

GAME_END_BEGIN_W

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

----------------------------

-- 0 => BEGIN

-- 1 => END

----------------------------

entity GAME_END_BEGIN_W is

PORT(

SEL_END_BEGIN : in STD_LOGIC;

CLR : in STD_LOGIC;

VIDON: in STD_LOGIC;

HC : in STD_LOGIC_VECTOR(10 downto 0);

VC : in STD_LOGIC_VECTOR(10 downto 0);

RED : out STD_LOGIC_VECTOR(3 downto 0);

GREEN : out STD_LOGIC_VECTOR(3 downto 0);

BLUE : out STD_LOGIC_VECTOR(3 downto 0)

);

end GAME_END_BEGIN_W;


architecture GAME_END_BEGIN_W of GAME_END_BEGIN_W is

SIGNAL R : STD_LOGIC_VECTOR(71 DOWNTO 0);

SIGNAL G : STD_LOGIC_VECTOR(71 DOWNTO 0);

SIGNAL B : STD_LOGIC_VECTOR(71 DOWNTO 0);

SIGNAL R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : STD_LOGIC_VECTOR(3 DOWNTO 0);

SIGNAL B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11 : STD_LOGIC_VECTOR(3 DOWNTO 0);

SIGNAL G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11 : STD_LOGIC_VECTOR(3 DOWNTO 0);

CONSTANT A : INTEGER := 10;

COMPONENT WORD_SELECTOR

PORT(

SEL_WORD : in std_logic_vector(7 downto 0);

R : in integer:= 0;

C : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0)

);
```

```
END COMPONENT;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;
begin
---------------------------------------------

UM1: CHAR_PLACER
PORT MAP(
MUX_IN => x"16",
R => 120,
C => 224-A,
HC => HC,
VC => VC,
RED => R1,
GREEN => G1,
BLUE => B1
);
UM2: CHAR_PLACER
PORT MAP(
MUX_IN => x"07",
R => 120,
C => 232-A,
HC => HC,
VC => VC,
RED => R2,
GREEN => G2,
BLUE => B2
);
UM3: CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => 120,
C => 240-A,
HC => HC,
VC => VC,
RED => R3,
GREEN => G3,
BLUE => B3
);
UM4: CHAR_PLACER
PORT MAP(
MUX_IN => x"02",
R => 120,
C => 248-A,
HC => HC,
VC => VC,
RED => R4,
GREEN => G4,
BLUE => B4
);
UM5: CHAR_PLACER
PORT MAP(
MUX_IN => x"0A",
R => 120,
C => 256-A,
HC => HC,
VC => VC,
RED => R5,
GREEN => G5,
BLUE => B5
);
UM6: CHAR_PLACER
PORT MAP(
MUX_IN => x"25",
R => 120,
C => 264-A,
HC => HC,
VC => VC,
RED => R6,
GREEN => G6,
BLUE => B6
);
UM7: CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => 120,
C => 272-A,
HC => HC,
VC => VC,
RED => R7,
GREEN => G7,
BLUE => B7
);

UM8: CHAR_PLACER
PORT MAP(
MUX_IN => x"25",
R => 120,
C => 280-A,
HC => HC,
VC => VC,
RED => R8,
GREEN => G8,
BLUE => B8
);
UM9: CHAR_PLACER
PORT MAP(
MUX_IN => x"0C",
R => 120,
C => 288-A,
HC => HC,
VC => VC,
RED => R9,
GREEN => G9,
BLUE => B9
);
UM10: CHAR_PLACER
PORT MAP(
MUX_IN => x"0E",
R => 120,
C => 296-A,
HC => HC,
VC => VC,
RED => R10,
GREEN => G10,
BLUE => B10
);

UM11: CHAR_PLACER
PORT MAP(
MUX_IN => x"0B",
R => 120,
C => 304-A,
HC => HC,
VC => VC,
RED => R11,
GREEN => G11,
BLUE => B11
);
U1: CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => 120,
C => 312-A,
HC => HC,
VC => VC,
RED => R(71 DOWNTO 68),
GREEN => G(71 DOWNTO 68),
BLUE => B(71 DOWNTO 68)
);
---------------------------------------------
U2: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"07",
R => 120,
C => 328-A,
HC => HC,
VC => VC,
RED => R(67 DOWNTO 64),
GREEN => G(67 DOWNTO 64),
BLUE => B(67 DOWNTO 64)
);

U3: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"0C",
R => 120,
C => 368-A,
HC => HC,
VC => VC,
RED => R(63 DOWNTO 60),
GREEN => G(63 DOWNTO 60),
BLUE => B(63 DOWNTO 60)
);

U4: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"0A",
R => 120,
C => 392-A,
HC => HC,
VC => VC,
RED => R(59 DOWNTO 56),
GREEN => G(59 DOWNTO 56),
BLUE => B(59 DOWNTO 56)
);

U5: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"03",
R => 120,
C => 424-A,
HC => HC,
VC => VC,
RED => R(55 DOWNTO 52),
GREEN => G(55 DOWNTO 52),
BLUE => B(55 DOWNTO 52)
);
-----------------------------------------------------
U6: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"05",
R => 360,
C => 240,
HC => HC,
VC => VC,
RED => R(51 DOWNTO 48),
GREEN => G(51 DOWNTO 48),
BLUE => B(51 DOWNTO 48)
);

U7: CHAR_PLACER
PORT MAP(
MUX_IN => x"28",
R => 360,
C => 288,
HC => HC,
VC => VC,
RED => R(47 DOWNTO 44),
GREEN => G(47 DOWNTO 44),
BLUE => B(47 DOWNTO 44)
```

```
);

U8: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"0B",
R => 360,
C => 304,
HC => HC,
VC => VC,
RED => R(43 DOWNTO 40),
GREEN => G(43 DOWNTO 40),
BLUE => B(43 DOWNTO 40)
);

U9: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"07",
R => 360,
C => 336,
HC => HC,
VC => VC,
RED => R(39 DOWNTO 36),
GREEN => G(39 DOWNTO 36),
BLUE => B(39 DOWNTO 36)
);

-----------------------------------------------------
U14: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"07",
R => 240,
C => 288,
HC => HC,
VC => VC,
RED => R(19 DOWNTO 16),
GREEN => G(19 DOWNTO 16),
BLUE => B(19 DOWNTO 16)
```

```
);

U15: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"12",
R => 240,
C => 328,
HC => HC,
VC => VC,
RED => R(15 DOWNTO 12),
GREEN => G(15 DOWNTO 12),
BLUE => B(15 DOWNTO 12)
);
--------------------------------
U16: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"04",
R => 360,
C => 256,
HC => HC,
VC => VC,
RED => R(11 DOWNTO 8),
GREEN => G(11 DOWNTO 8),
BLUE => B(11 DOWNTO 8)
);

U17: WORD_SELECTOR
PORT MAP(
SEL_WORD => x"0B",
R => 360,
C => 312,
HC => HC,
VC => VC,
RED => R(7 DOWNTO 4),
GREEN => G(7 DOWNTO 4),
BLUE => B(7 DOWNTO 4)
);
```

```
U18: WORD_SELECTOR

PORT MAP(

SEL_WORD => x"01",

R => 360,

C => 344,

HC => HC,

VC => VC,

RED => R(3 DOWNTO 0),

GREEN => G(3 DOWNTO 0),

BLUE => B(3 DOWNTO 0)

);

-------------------------------------

PROCESS(VIDON,SEL_END_BEGIN,R,G,B)

BEGIN

CASE SEL_END_BEGIN IS

WHEN '0' => --- BEGIN SCREEN

IF VIDON = '1' THEN

RED <= R(71 DOWNTO 68) OR R(67 DOWNTO 64) OR R(63 DOWNTO 60) OR R(59 DOWNTO 56) OR R(55 DOWNTO
52) OR R(51 DOWNTO 48) OR R(47 DOWNTO 44) OR R(43 DOWNTO 40) OR R(39 DOWNTO 36)

OR R1 OR R2 OR R3 OR R4 OR R5 OR R6 OR R7 OR R8 OR R9 OR R10 OR R11;

GREEN <= X"0" OR G(67 DOWNTO 64) OR G(63 DOWNTO 60) OR G(59 DOWNTO 56) OR G(55 DOWNTO 52) OR G(51
DOWNTO 48) OR G(47 DOWNTO 44) OR G(43 DOWNTO 40) OR G(39 DOWNTO 36)

OR G1 OR G2 OR G3 OR G4 OR G5 OR G6 OR G7 OR G8 OR X"0" OR X"0" OR X"0";

BLUE <= B(71 DOWNTO 68) OR B(67 DOWNTO 64) OR B(63 DOWNTO 60) OR B(59 DOWNTO 56) OR B(55 DOWNTO
52) OR B(51 DOWNTO 48) OR B(47 DOWNTO 44) OR B(43 DOWNTO 40) OR B(39 DOWNTO 36)

OR X"0" OR X"0" OR X"0" OR X"0" OR X"0" OR B6 OR B7 OR B8 OR B9 OR B10 OR B11;

ELSE

RED <= "0000";

GREEN <= "0000";

BLUE <= "0000";

END IF;


WHEN '1' =>  ----ENDING SCREEN

IF VIDON = '1' THEN

RED <= R(19 DOWNTO 16) OR R(15 DOWNTO 12) OR R(11 DOWNTO 8) OR R(7 DOWNTO 4) OR R(3 DOWNTO 0);

GREEN <= G(19 DOWNTO 16) OR G(15 DOWNTO 12) OR G(11 DOWNTO 8) OR G(7 DOWNTO 4) OR G(3 DOWNTO 0);

BLUE <= B(19 DOWNTO 16) OR B(15 DOWNTO 12) OR B(11 DOWNTO 8) OR B(7 DOWNTO 4) OR B(3 DOWNTO 0);
```

```
ELSE
RED <= "0000";
GREEN <= "0000";
BLUE <= "0000";
END IF;

END CASE;
END PROCESS;


end GAME_END_BEGIN_W;
```

<u>WORD_SELECTOR</u>

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
------------------------
--00 => CORRECT
--01 => PLAYING
--02 => MEMORY
--03 => TARHAN
--04 => THANKS
--05 => PRESS
--06 => WRONG
--07 => GAME
--08 => TURN
--09 => DOWN
--0A => EFE
--0B => FOR
--0C => BY
--0D => P1
--0E => P2
--0F => UP
--10 => SINGLEPLAYER
--11 => MULTIPLAYER
--12 => OVER
```

```
--13 => NEXT
------------------------
entity WORD_SELECTOR is
PORT(
SEL_WORD : in std_logic_vector(7 downto 0);
R : in integer:= 0;
C : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
end WORD_SELECTOR;

architecture WORD_SELECTOR of WORD_SELECTOR is
SIGNAL R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16,R17,R18,R19,R20 : STD_LOGIC_VECTOR(3
DOWNTO 0);
SIGNAL B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,B16,B17,B18,B19,B20 : STD_LOGIC_VECTOR(3
DOWNTO 0);
SIGNAL G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15,G16,G17,G18,G19,G20 : STD_LOGIC_VECTOR(3
DOWNTO 0);
COMPONENT CORRECT
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT PLAYING
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
```

```
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT MEMORY
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT TARHAN
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT THANKS
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
```

```
);
END COMPONENT;
COMPONENT PRESS
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT WRONG
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT GAME
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT TURN
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT W_DOWN
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT EFE
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT W_FOR
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
```

```
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT W_BY
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
COMPONENT W_P1
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;

COMPONENT W_P2
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
```

```
);
END COMPONENT;
COMPONENT W_UP
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;

COMPONENT SINGLEPLAYER
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;

COMPONENT MULTIPLAYER
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
```

```
COMPONENT OVER
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;

COMPONENT W_NEXT
PORT(
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0)
);
END COMPONENT;
begin

CU1: CORRECT
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R1,
GREEN => G1,
BLUE => B1
);
```

```
CU2: PLAYING
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R2,
GREEN => G2,
BLUE => B2
);

CU3: MEMORY
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R3,
GREEN => G3,
BLUE => B3
);

CU4: TARHAN
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R4,
GREEN => G4,
BLUE => B4
);

CU5: THANKS
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R5,
GREEN => G5,
BLUE => B5
);

CU6: PRESS
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R6,
GREEN => G6,
BLUE => B6
);

CU7: WRONG
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R7,
GREEN => G7,
BLUE => B7
);

CU8: GAME
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R8,
GREEN => G8,
BLUE => B8
);

CU9: TURN
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R9,
GREEN => G9,
BLUE => B9
);
CU10: W_DOWN
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R10,
GREEN => G10,
BLUE => B10
);

CU11: EFE
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R11,
GREEN => G11,
BLUE => B11
);

CU12: W_FOR
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R12,
GREEN => G12,
BLUE => B12
);

CU13: W_BY
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R13,
GREEN => G13,
BLUE => B13
);
CU14: W_P1
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R14,
GREEN => G14,
BLUE => B14
);

CU15: W_P2
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R15,
GREEN => G15,
BLUE => B15
);

CU16: W_UP
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R16,
GREEN => G16,
BLUE => B16
);

CU17: SINGLEPLAYER
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R17,
GREEN => G17,
BLUE => B17
);

CU18: MULTIPLAYER
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R18,
GREEN => G18,
BLUE => B18
);
```

```
CU19: OVER
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R19,
GREEN => G19,
BLUE => B19
);
CU20: W_NEXT
PORT MAP(
Row => R,
Col => C,
HC => HC,
VC => VC,
RED => R20,
GREEN => G20,
BLUE => B20
);
----------------------------------------------------------CASE---------------------------


PROCESS(SEL_WORD,
R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,R12,R13,R14,R15,R16,R17,R18,R19,R20,
B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,B16,B17,B18,B19,B20,
G1,G2,G3,G4,G5,G6,G7,G8,G9,G10,G11,G12,G13,G14,G15,G16,G17,G18,G19,G20)
BEGIN
CASE SEL_WORD IS

WHEN X"00" => -- CORRECT
RED <= "0000"; GREEN <= G1; BLUE <= "0000";
WHEN X"01" => --PLAYING
RED <= R2; GREEN <= "0000"; BLUE <= B2;
WHEN X"02" => --MEMORY
RED <= R3; GREEN <= G3; BLUE <= "0000";
WHEN X"03" => --TARHAN
RED <= R4; GREEN <= G4; BLUE <= B4;
```

```
WHEN X"04" => --THANKS

RED <= R5; GREEN <= "0000"; BLUE <= "0000";

WHEN X"05" => --PRESS

RED <= "0000"; GREEN <= G6; BLUE <= "0000";

WHEN X"06" => --WRONG

RED <= R7; GREEN <= "0000"; BLUE <= "0000";

WHEN X"07" => --GAME

RED <= R8; GREEN <= G8; BLUE <= B8;

WHEN X"08" => --TURN

RED <= R9; GREEN <= G9; BLUE <= B9;

WHEN X"09" => --DOWN

RED <= R10; GREEN <= G10; BLUE <= B10;

WHEN X"0A" => --EFE

RED <= "0000"; GREEN <= G11; BLUE <= B11;

WHEN X"0B" => --FOR

RED <= R12; GREEN <= G12; BLUE <= B12;

WHEN X"0C" => --BY

RED <= R13; GREEN <= "0000"; BLUE <= "0000";

WHEN X"0D" => --P1

RED <= R14; GREEN <= "0000"; BLUE <= "0000";

WHEN X"0E" => --P2

RED <= "0000"; GREEN <= "0000"; BLUE <= B15;

WHEN X"0F" => --UP

RED <= R16; GREEN <= G16; BLUE <= B16;

WHEN X"10" => --SINGLEPLAYER

RED <= R17; GREEN <= G17; BLUE <= B17;

WHEN X"11" => --MULTPLAYER

RED <= R18; GREEN <= G18; BLUE <= B18;

WHEN X"12" => --OVER

RED <= R19; GREEN <= G19; BLUE <= "0000";

WHEN X"13" => --NEXT

RED <= R20; GREEN <= G20; BLUE <= B20;

WHEN OTHERS => NULL;

END CASE;

END PROCESS;


end WORD_SELECTOR;
```

CORRECT

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CORRECT is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end CORRECT;


architecture CORRECT of CORRECT is

signal RE : std_logic_vector(27 downto 0);

signal BL : std_logic_vector(27 downto 0);

signal GR : std_logic_vector(27 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;

signal C5 : integer := Col + 4*X;

signal C6 : integer := Col + 5*X;

signal C7 : integer := Col + 6*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);
```

```
BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"02",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0E",

R => Row,

C => C2,

HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)

);


CU3 : CHAR_PLACER

PORT MAP(

MUX_IN => x"11",

R => Row,

C => C3,

HC => HC,

VC => VC,

RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

CU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C5,
HC => HC,
VC => VC,
RED => RE(7 downto 4),
GREEN => GR(7 downto 4),
BLUE => BL(7 downto 4)
);

CU6 : CHAR_PLACER
PORT MAP(
MUX_IN => x"02",
R => Row,
C => C6,
HC => HC,
VC => VC,
RED => RE(3 downto 0),
GREEN => GR(3 downto 0),
BLUE => BL(3 downto 0)

);


CU7 : CHAR_PLACER

PORT MAP(

MUX_IN => x"13",

R => Row,

C => C7,

HC => HC,

VC => VC,

RED => RE(27 downto 24),

GREEN => GR(27 downto 24),

BLUE => BL(27 downto 24)

);


RED <= RE(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11
DOWNTO 8) OR RE(7 DOWNTO 4) OR RE(3 DOWNTO 0);

GREEN <= GR(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11
DOWNTO 8) OR GR(7 DOWNTO 4) OR GR(3 DOWNTO 0);

BLUE <= BL(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11
DOWNTO 8) OR BL(7 DOWNTO 4) OR BL(3 DOWNTO 0);


end CORRECT;
```

<u>PLAYING</u>

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PLAYING is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end PLAYING;
```

```
architecture PLAYING of PLAYING is
signal RE : std_logic_vector(27 downto 0);
signal BL : std_logic_vector(27 downto 0);
signal GR : std_logic_vector(27 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;
signal C6 : integer := Col + 5*X;
signal C7 : integer := Col + 6*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0B",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"18",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);
```

```
CU5 : CHAR_PLACER

PORT MAP(

MUX_IN => x"08",

R => Row,

C => C5,

HC => HC,

VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);


CU6 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0D",

R => Row,

C => C6,

HC => HC,

VC => VC,

RED => RE(3 downto 0),

GREEN => GR(3 downto 0),

BLUE => BL(3 downto 0)

);


CU7 : CHAR_PLACER

PORT MAP(

MUX_IN => x"06",

R => Row,

C => C7,

HC => HC,

VC => VC,

RED => RE(27 downto 24),

GREEN => GR(27 downto 24),

BLUE => BL(27 downto 24)

);
```

```
RED <= RE(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11
DOWNTO 8) OR RE(7 DOWNTO 4) OR RE(3 DOWNTO 0);

GREEN <= GR(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11
DOWNTO 8) OR GR(7 DOWNTO 4) OR GR(3 DOWNTO 0);

BLUE <= BL(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11
DOWNTO 8) OR BL(7 DOWNTO 4) OR BL(3 DOWNTO 0);


end PLAYING;
```

MEMORY

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MEMORY is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end MEMORY;


architecture MEMORY of MEMORY is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;

signal C5 : integer := Col + 4*X;

signal C6 : integer := Col + 5*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
```

```
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0C",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0C",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0E",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

CU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C5,
HC => HC,
VC => VC,
RED => RE(7 downto 4),
GREEN => GR(7 downto 4),
BLUE => BL(7 downto 4)
);

CU6 : CHAR_PLACER
PORT MAP(
MUX_IN => x"18",
R => Row,

C => C6,

HC => HC,

VC => VC,

RED => RE(3 downto 0),

GREEN => GR(3 downto 0),

BLUE => BL(3 downto 0)

);


RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7
DOWNTO 4) OR RE(3 DOWNTO 0);

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7
DOWNTO 4) OR GR(3 DOWNTO 0);

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7
DOWNTO 4) OR BL(3 DOWNTO 0);


end MEMORY;
```

TARHAN

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TARHAN is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end TARHAN;


architecture TARHAN of TARHAN is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;
signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;

signal C5 : integer := Col + 4*X;

signal C6 : integer := Col + 5*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"13",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"00",

R => Row,

C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"07",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

CU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => Row,
C => C5,
HC => HC,
VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);


CU6 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0D",

R => Row,

C => C6,

HC => HC,

VC => VC,

RED => RE(3 downto 0),

GREEN => GR(3 downto 0),

BLUE => BL(3 downto 0)

);


RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7
DOWNTO 4) OR RE(3 DOWNTO 0);

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7
DOWNTO 4) OR GR(3 DOWNTO 0);

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7
DOWNTO 4) OR BL(3 DOWNTO 0);


end TARHAN;
```

THANKS

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity THANKS is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);
```

```
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end THANKS;

architecture THANKS of THANKS is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;
signal C6 : integer := Col + 5*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"13",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"07",

R => Row,

C => C2,

HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)

);


CU3 : CHAR_PLACER

PORT MAP(

MUX_IN => x"00",

R => Row,

C => C3,

HC => HC,

VC => VC,

RED => RE(15 downto 12),

GREEN => GR(15 downto 12),

BLUE => BL(15 downto 12)

);


CU4 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0D",

R => Row,

C => C4,

HC => HC,

VC => VC,

RED => RE(11 downto 8),

GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)

);


CU5 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0A",

R => Row,

C => C5,

HC => HC,

VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);


CU6 : CHAR_PLACER

PORT MAP(

MUX_IN => x"12",

R => Row,

C => C6,

HC => HC,

VC => VC,

RED => RE(3 downto 0),

GREEN => GR(3 downto 0),

BLUE => BL(3 downto 0)

);


RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7
DOWNTO 4) OR RE(3 DOWNTO 0);

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7
DOWNTO 4) OR GR(3 DOWNTO 0);

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7
DOWNTO 4) OR BL(3 DOWNTO 0);


end THANKS;
```

PRESS

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PRESS is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end PRESS;

architecture PRESS of PRESS is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin
```

```
CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
```

```
PORT MAP(

MUX_IN => x"12",

R => Row,

C => C4,

HC => HC,

VC => VC,

RED => RE(11 downto 8),

GREEN => GR(11 downto 8),

BLUE => BL(11 downto 8)

);


CU5 : CHAR_PLACER

PORT MAP(

MUX_IN => x"12",

R => Row,

C => C5,

HC => HC,

VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7
DOWNTO 4) ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7
DOWNTO 4) ;

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7
DOWNTO 4) ;


end PRESS;
```

WRONG

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity WRONG is

Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end WRONG;

architecture WRONG of WRONG is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"16",
R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"11",

R => Row,

C => C2,

HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)

);


CU3 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0E",

R => Row,

C => C3,

HC => HC,

VC => VC,

RED => RE(15 downto 12),

GREEN => GR(15 downto 12),

BLUE => BL(15 downto 12)

);


CU4 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0D",

R => Row,
C => C4,

HC => HC,

VC => VC,

RED => RE(11 downto 8),

GREEN => GR(11 downto 8),

BLUE => BL(11 downto 8)

);


CU5 : CHAR_PLACER

PORT MAP(

MUX_IN => x"06",

R => Row,

C => C5,

HC => HC,

VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7
DOWNTO 4) ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7
DOWNTO 4) ;

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7
DOWNTO 4) ;


end WRONG;
```

<u>GAME</u>

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity GAME is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);
```

```
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end GAME;

architecture GAME of GAME is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"06",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0C",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) ;

end GAME;
```

TURN

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity TURN is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end TURN;

architecture TURN of TURN is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
```

```
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"13",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"14",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",

R => Row,

C => C3,

HC => HC,

VC => VC,

RED => RE(15 downto 12),

GREEN => GR(15 downto 12),

BLUE => BL(15 downto 12)

);


CU4 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0D",

R => Row,

C => C4,

HC => HC,

VC => VC,

RED => RE(11 downto 8),

GREEN => GR(11 downto 8),

BLUE => BL(11 downto 8)

);


RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) ;

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) ;


end TURN;
```

W_DOWN

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity W_DOWN is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);
```

```
RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end W_DOWN;


architecture W_DOWN of W_DOWN is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"03",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0E",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"16",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0D",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) ;

end W_DOWN;
```

EFE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity EFE is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end EFE;

architecture EFE of EFE is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
```

```
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"05",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C3,

HC => HC,

VC => VC,

RED => RE(15 downto 12),

GREEN => GR(15 downto 12),

BLUE => BL(15 downto 12)

);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12);

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12);


end EFE;
```

W_FOR

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity W_FOR is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end W_FOR;


architecture W_FOR of W_FOR is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;
```

```
COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"05",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0E",

R => Row,

C => C2,

HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12);
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12);

end W_FOR;
```

W_BY

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity W_BY is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end W_BY;

architecture W_BY of W_BY is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"01",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"18",

R => Row,

C => C2,
HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)

);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16)  ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) ;

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) ;


end W_BY;
```

W_P1

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity W_P1 is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end W_P1;


architecture W_P1 of W_P1 is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;


COMPONENT CHAR_PLACER
```

```
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"1A",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);
```

```
RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16)  ;

GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) ;

BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) ;


end W_P1;
```

<u>W_P2</u>

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity W_P2 is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end W_P2;


architecture W_P2 of W_P2 is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"1B",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16)  ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) ;

end W_P2;
```

W_UP

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity W_UP is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end W_UP;


architecture W_UP of W_UP is

signal RE : std_logic_vector(23 downto 0);

signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin
```

```
CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"14",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);



RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16)  ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) ;


end W_UP;
```

SINGLEPLAYER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SINGLEPLAYER is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end SINGLEPLAYER;

architecture SINGLEPLAYER of SINGLEPLAYER is
signal RE : std_logic_vector(47 downto 0);
signal BL : std_logic_vector(47 downto 0);
signal GR : std_logic_vector(47 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;
signal C6 : integer := Col + 5*X;
signal C7 : integer := Col + 6*X;
signal C8 : integer := Col + 7*X;
signal C9 : integer := Col + 8*X;
signal C10 : integer := Col + 9*X;
signal C11 : integer := Col + 10*X;
signal C12 : integer := Col + 11*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;
```

```
begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"12",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(47 downto 44),
GREEN => GR(47 downto 44),
BLUE => BL(47 downto 44)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"08",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(43 downto 40),
GREEN => GR(43 downto 40),
BLUE => BL(43 downto 40)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0D",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(39 downto 36),
GREEN => GR(39 downto 36),
BLUE => BL(39 downto 36)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"06",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(35 downto 32),
GREEN => GR(35 downto 32),
BLUE => BL(35 downto 32)
);

CU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0B",
R => Row,
C => C5,
HC => HC,
VC => VC,
RED => RE(31 downto 28),
GREEN => GR(31 downto 28),
BLUE => BL(31 downto 28)
);

CU6 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C6,
HC => HC,
VC => VC,
RED => RE(27 downto 24),
GREEN => GR(27 downto 24),
BLUE => BL(27 downto 24)
);
```

```
CU7 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0F",
R => Row,
C => C7,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU8 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0B",
R => Row,
C => C8,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU9 : CHAR_PLACER
PORT MAP(
MUX_IN => x"00",
R => Row,
C => C9,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);
```

```
CU10 : CHAR_PLACER
PORT MAP(
MUX_IN => x"18",
R => Row,
C => C10,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

CU11 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C11,
HC => HC,
VC => VC,
RED => RE(7 downto 4),
GREEN => GR(7 downto 4),
BLUE => BL(7 downto 4)
);

CU12 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C12,
HC => HC,
VC => VC,
RED => RE(3 downto 0),
GREEN => GR(3 downto 0),
BLUE => BL(3 downto 0)
);
```

```
RED <= RE(47 DOWNTO 44) OR RE(43 DOWNTO 40) OR RE(39 DOWNTO 36) OR RE(35 DOWNTO 32) OR RE(31
DOWNTO 28) OR RE(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR

RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7 DOWNTO 4) OR RE(3 DOWNTO 0);


GREEN <= GR(47 DOWNTO 44) OR GR(43 DOWNTO 40) OR GR(39 DOWNTO 36) OR GR(35 DOWNTO 32) OR GR(31
DOWNTO 28) OR GR(27 DOWNTO 24) OR GR(23 DOWNTO 20) OR

GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7 DOWNTO 4) OR GR(3 DOWNTO 0);


BLUE  <= BL(47 DOWNTO 44) OR BL(43 DOWNTO 40) OR BL(39 DOWNTO 36) OR BL(35 DOWNTO 32) OR BL(31
DOWNTO 28) OR BL(27 DOWNTO 24) OR BL(23 DOWNTO 20) OR

BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7 DOWNTO 4) OR BL(3 DOWNTO 0);


END SINGLEPLAYER;
```

<u>**MULTIPLAYER**</u>

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MULTIPLAYER is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end MULTIPLAYER;


architecture MULTIPLAYER of MULTIPLAYER is

signal RE : std_logic_vector(47 downto 0);

signal BL : std_logic_vector(47 downto 0);

signal GR : std_logic_vector(47 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;
signal C5 : integer := Col + 4*X;

signal C6 : integer := Col + 5*X;

signal C7 : integer := Col + 6*X;

signal C8 : integer := Col + 7*X;

signal C9 : integer := Col + 8*X;

signal C10 : integer := Col + 9*X;

signal C11 : integer := Col + 10*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0C",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(47 downto 44),

GREEN => GR(47 downto 44),

BLUE => BL(47 downto 44)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"14",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(43 downto 40),
GREEN => GR(43 downto 40),
BLUE => BL(43 downto 40)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0B",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(39 downto 36),
GREEN => GR(39 downto 36),
BLUE => BL(39 downto 36)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"13",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(35 downto 32),
GREEN => GR(35 downto 32),
BLUE => BL(35 downto 32)
);

CU5 : CHAR_PLACER
PORT MAP(
MUX_IN => x"08",
R => Row,
C => C5,

HC => HC,

VC => VC,

RED => RE(31 downto 28),

GREEN => GR(31 downto 28),

BLUE => BL(31 downto 28)

);


CU6 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0F",

R => Row,

C => C6,

HC => HC,

VC => VC,

RED => RE(27 downto 24),

GREEN => GR(27 downto 24),

BLUE => BL(27 downto 24)

);


CU7 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0B",

R => Row,

C => C7,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU8 : CHAR_PLACER

PORT MAP(

MUX_IN => x"00",

R => Row,

C => C8,
HC => HC,

VC => VC,

RED => RE(19 downto 16),

GREEN => GR(19 downto 16),

BLUE => BL(19 downto 16)

);


CU9 : CHAR_PLACER

PORT MAP(

MUX_IN => x"18",

R => Row,

C => C9,

HC => HC,

VC => VC,

RED => RE(15 downto 12),

GREEN => GR(15 downto 12),

BLUE => BL(15 downto 12)

);


CU10 : CHAR_PLACER

PORT MAP(

MUX_IN => x"04",

R => Row,

C => C10,

HC => HC,

VC => VC,

RED => RE(11 downto 8),

GREEN => GR(11 downto 8),

BLUE => BL(11 downto 8)

);


CU11 : CHAR_PLACER

PORT MAP(

MUX_IN => x"11",

R => Row,

C => C11,

HC => HC,
VC => VC,

RED => RE(7 downto 4),

GREEN => GR(7 downto 4),

BLUE => BL(7 downto 4)

);




RED <= RE(47 DOWNTO 44) OR RE(43 DOWNTO 40) OR RE(39 DOWNTO 36) OR RE(35 DOWNTO 32) OR RE(31
DOWNTO 28) OR RE(27 DOWNTO 24) OR RE(23 DOWNTO 20) OR

RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) OR RE(7 DOWNTO 4);


GREEN <= GR(47 DOWNTO 44) OR GR(43 DOWNTO 40) OR GR(39 DOWNTO 36) OR GR(35 DOWNTO 32) OR GR(31
DOWNTO 28) OR GR(27 DOWNTO 24) OR GR(23 DOWNTO 20) OR

GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) OR GR(7 DOWNTO 4);


BLUE  <= BL(47 DOWNTO 44) OR BL(43 DOWNTO 40) OR BL(39 DOWNTO 36) OR BL(35 DOWNTO 32) OR BL(31
DOWNTO 28) OR BL(27 DOWNTO 24) OR BL(23 DOWNTO 20) OR

BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) OR BL(7 DOWNTO 4);


END MULTIPLAYER;
```

OVER

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OVER is

Port (

Row : in integer:= 0;

Col : in integer:= 0;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

end OVER;


architecture OVER of OVER is

signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);

signal GR : std_logic_vector(23 downto 0);

signal X: integer := 8;

signal C1 :integer := Col;

signal C2 : integer := Col +X;

signal C3 : integer := Col + 2*X;

signal C4 :integer := Col + 3*X;


COMPONENT CHAR_PLACER

PORT(

MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);

R : in integer;

C : in integer;

HC : in STD_LOGIC_VECTOR (10 downto 0);

VC : in STD_LOGIC_VECTOR (10 downto 0);

RED : out STD_LOGIC_VECTOR (3 downto 0);

GREEN : out STD_LOGIC_VECTOR (3 downto 0);

BLUE : out STD_LOGIC_VECTOR (3 downto 0));

END COMPONENT;


begin


CU1 : CHAR_PLACER

PORT MAP(

MUX_IN => x"0E",

R => Row,

C => C1,

HC => HC,

VC => VC,

RED => RE(23 downto 20),

GREEN => GR(23 downto 20),

BLUE => BL(23 downto 20)

);


CU2 : CHAR_PLACER

PORT MAP(

MUX_IN => x"15",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"11",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) ;
```

```
end OVER;
```

W_NEXT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity W_NEXT is
Port (
Row : in integer:= 0;
Col : in integer:= 0;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
end W_NEXT;

architecture W_NEXT of W_NEXT is
signal RE : std_logic_vector(23 downto 0);
signal BL : std_logic_vector(23 downto 0);
signal GR : std_logic_vector(23 downto 0);
signal X: integer := 8;
signal C1 :integer := Col;
signal C2 : integer := Col +X;
signal C3 : integer := Col + 2*X;
signal C4 :integer := Col + 3*X;

COMPONENT CHAR_PLACER
PORT(
MUX_IN: in STD_LOGIC_VECTOR(7 downto 0);
R : in integer;
C : in integer;
HC : in STD_LOGIC_VECTOR (10 downto 0);
VC : in STD_LOGIC_VECTOR (10 downto 0);
RED : out STD_LOGIC_VECTOR (3 downto 0);
GREEN : out STD_LOGIC_VECTOR (3 downto 0);
BLUE : out STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;
```

```
begin

CU1 : CHAR_PLACER
PORT MAP(
MUX_IN => x"0D",
R => Row,
C => C1,
HC => HC,
VC => VC,
RED => RE(23 downto 20),
GREEN => GR(23 downto 20),
BLUE => BL(23 downto 20)
);

CU2 : CHAR_PLACER
PORT MAP(
MUX_IN => x"04",
R => Row,
C => C2,
HC => HC,
VC => VC,
RED => RE(19 downto 16),
GREEN => GR(19 downto 16),
BLUE => BL(19 downto 16)
);

CU3 : CHAR_PLACER
PORT MAP(
MUX_IN => x"17",
R => Row,
C => C3,
HC => HC,
VC => VC,
RED => RE(15 downto 12),
GREEN => GR(15 downto 12),
BLUE => BL(15 downto 12)
);

CU4 : CHAR_PLACER
PORT MAP(
MUX_IN => x"13",
R => Row,
C => C4,
HC => HC,
VC => VC,
RED => RE(11 downto 8),
GREEN => GR(11 downto 8),
BLUE => BL(11 downto 8)
);

RED <= RE(23 DOWNTO 20) OR RE(19 DOWNTO 16) OR RE(15 DOWNTO 12) OR RE(11 DOWNTO 8) ;
GREEN <= GR(23 DOWNTO 20) OR GR(19 DOWNTO 16) OR GR(15 DOWNTO 12) OR GR(11 DOWNTO 8) ;
BLUE <= BL(23 DOWNTO 20) OR BL(19 DOWNTO 16) OR BL(15 DOWNTO 12) OR BL(11 DOWNTO 8) ;

end W_NEXT;
```